Reliable Communication over Channels with Insertions, Deletions, and Substitutions

Matthew C. Davey and David J. C. MacKay

***Abstract*—A new block code is introduced which is capable of correcting multiple insertion, deletion, and substitution errors. The code consists of nonlinear inner codes, which we call "watermark" codes, concatenated with low-density parity-check codes over nonbinary fields. The inner code allows probabilistic resynchronization and provides soft outputs for the outer decoder, which then completes decoding. We present codes of rate 0.7 and transmitted length 5000 bits that can correct 30 insertion/deletion errors per block. We also present codes of rate $3/14$ and length 4600 bits that can correct 450 insertion/deletion errors per block.**

***Index Terms*—Channel coding, concatenated coding, error correction, hidden Markov models (HMMs), insertion/deletion channels, synchronization.**

I. Introduction

MOST block error-correcting codes are designed to correct *substitution* errors, in which the transmitted symbol is replaced by a different received symbol. The decoder must be told (or be able to identify) the block boundaries. Thus, ensuring synchronization of the encoder and decoder is a crucial element of communication systems.

For channels that make *synchronization* errors, i.e., insertions and deletions of symbols, alternative coding methods must be used. We distinguish between two types of codes for synchronization errors. First, there are codes that *detect* loss of synchronization and regain synchronization. Such codes do not correct errors in blocks corrupted by insertions and deletions. Second, there are codes that can correct synchronization errors *and* recover the transmitted block even when it is corrupted by insertion and deletion errors. Our codes are of this second type.

A. Codes that Regain Synchronization

A comma-free code has the property that if $x = x_1 x_2 \cdots x_n$ and $y = y_1 y_2 \cdots y_n$ are codewords, then none of the overlaps

$$x_i \cdots x_n y_1 \cdots y_{i-1}, \qquad 1 < i \leq n$$

is a valid codeword. Hence, if the incorrect block alignment is chosen (assuming no insertion, deletion, or substitution errors), the data do not form valid codewords. Comma-free codes allow a receiver to resynchronize following a synchronization error with a delay of at most two blocks of data, although blocks corrupted by insertions and deletions are not recoverable.

Comma-free codes were originally proposed by Crick, Griffith, and Orgel in 1957 [1] as a possible encoding of protein sequences in DNA. They were introduced to the coding community by Golomb, Gordon, and Welch [2] the following year.

Some comma-free codes can also correct substitution errors. Hatcher [3] constructed comma-free codes that could resynchronize and correct a limited number of substitution errors. Stiffler [4] and Tavares and Fukada [5] proposed adding a constant vector to binary cyclic codes (a *coset* construction) to create comma-free codes with the error-correction capability of cyclic codes. None of these methods allow error correction in blocks with insertion or deletion errors.

B. Codes that Correct Synchronization Errors

The first codes capable of correcting a codeword corrupted by insertions and deletions were introduced by Sellers [6]. A synchronizing *marker* sequence (e.g. "001") was inserted into an outer burst-error-correcting code at periodic intervals. By searching for the markers in their expected positions, the decoder could detect and subsequently correct single insertions or deletions between successive markers. Multiple synchronization errors could be corrected by using longer markers, at the expense of added redundancy.

Most subsequent work on insertion/deletion correcting codes has focused on the number-theoretic constructions employed by Levenshtein [7]–[11]. Levenshtein proved that these codes could correct single insertions and deletions under the assumption that the codeword boundaries were given, and gave a decoding algorithm.

Levenshtein also calculated asymptotic bounds on the number of codewords in a code capable of correcting up to s insertion/deletion errors as the code length $n \to \infty$. Ullman [12] produced bounds on the rate of codes capable of correcting $n\alpha$ insertion/deletion errors in a block of length n, as $n \to \infty$. In an unpublished 1961 report, Gallager [32] suggested adding a pseudorandom string to convolutional codes and using sequential decoding to correct synchronization and substitution errors. He found a computational cutoff rate above which the expected cost of sequential decoding of these codes is infinite. Zigangirov [13] found lower bounds for the capacity of a more general insetion/deletion channel using tree codes. These bounds are illustrated in Fig. 15. Our codes can achieve a small probability of error at rates above the lower bounds of Ullman and Levenshtein, and above the computational cutoff rate.

Manuscript received December 15, 1999; revised September 14, 2000. This work was supported by the Gatsby Foundation. The work of M. C. Davey was supported by the Schiff Foundation and a Gardiner scholarship. The material in this paper was presented at the IEEE International Symposium on Information Theory, Sorrento, Italy, June 2000.

M. C. Davey is with the Cooperative Research Centre for Enterprise Distributed Systems, Brisbane, Qld. 4001, Australia (e-mail: mcdavey@dstc.edu.au).

D. J. C. MacKay is with the Cavendish Laboratory, Cambridge CB3 0ME, U.K. (e-mail: mackay@mrao.cam.ac.uk).

Communicated by E. D. Forney, Guest Editor.

Publisher Item Identifier S 0018-9448(01)00732-5.

Recently, Schulman and Zuckerman [15] described how to construct concatenated codes that are asymptotically good for channels with insertions, deletions, and transpositions, and that are polynomial-time encodeable and decodeable. The inner code of their concatenated code is constructed and decoded by brute force, and has a block length that scales as $\log N$, where N is the outer code length. While it is true that these codes can be encoded and decoded in time polynomial in N, they appear to be more of theoretical than of practical importance. No practical performance curves are given by Schulman and Zuckerman, so we have not been able to compare the codes we present with theirs. The decoding complexity of our codes is of order $N \log N$.

C. Outline of Paper

In Section II we describe a new construction of codes for the correction of synchronization errors. An inner code is used to infer the position of insertion/deletion errors and an outer code corrects errors caused by substitutions, deletions, and misidentified synchronization errors. We use a low-density parity-check code as the outer code because these codes are able to make use of soft inputs (likelihood ratios), and because low-density parity-check codes appear to be the best known practical codes for the rates and block lengths needed here [16]–[18]. We call the resynchronizing inner codes "watermark codes." Watermark codes are suitable for a broad class of insertion/deletion channels. A simple channel model is introduced in Section III.

A detailed description of the encoding and decoding of watermark codes is given in Sections IV and V, respectively.

The empirical performance of watermark codes concatenated with low-density parity-check codes is shown in Section VI. We present codes with overall rate 0.7 and block length 5000 bits that achieve a block-error rate of 10^{-3} for received blocks corrupted by an *average* of 14 synchronization and 14 substitution errors per block. Rate-$3/14$ codes of length 4600 bits are shown to be able to correct over 450 synchronization errors per block, achieving a block-error rate of 10^{-3}.

Section VII discusses efficient implementation of the decoding algorithm and the design of good watermark codes. Finally, Section VIII outlines several promising areas for future work. An Appendix examines the synchronization performance of watermark codes and presents bounds on the capacity of insertion/deletion channels.

II. Outline of Watermark Codes for Synchronization

We break the decoding problem into two parts. First, we identify, as best we can, where the insertions and deletions occurred. We do this using an inner code that allows us to infer the position of synchronization errors. The output of this inner code is expected to have several residual errors from misidentified synchronization errors and the uncorrected substitution errors. Second, an outer code corrects the remaining errors.

The key idea in the inner code is to provide a carrier signal or "watermark" for the decoder. If there are synchronization errors then the decoder identifies discontinuities in the carrier signal and deduces the presence of the errors. This idea is similar to that of marker codes [6], [19] which insert a known pattern at regular intervals. Using marker codes, synchronization is regained by inserting or deleting symbols (as appropriate) in sections that suffer from synchronization errors. This resynchronization procedure typically produces a burst of substitution errors. In a watermark code, the "marker" is distributed uniformly throughout the block and synchronization is recovered using the sum–product algorithm. Marker codes can be seen as a special case of watermark codes.

First imagine that the receiver knows the message being transmitted. In this case, the problem of identifying where the insertions and deletions occurred is easier to formulate. This problem is closely related to that of generating alignments of biological sequences and, more generally, to the notions of string similarity and edit distance [20]–[23]. We can solve the problem optimally and efficiently using the sum–product algorithm.

As an example, take the following transmitted and received strings, where we label the positions with letters:

```
Index:       ABCDEFGHIJKLMNOPQRSTUVWXYZ
Transmitted: 01001010011100101111010001
Received:    01001010111011011111010001
```

In this case, the receiver might conclude that there has been a deletion at position "H" or "I," a substitution in position "N," and an insertion somewhere between positions "Q" and "U" (of course, we must assign probabilities to insertion, deletion, and substitution events to quantify this conclusion).

Now, if the sender were to make a few substitutions to the agreed string before transmission then the receiver could still retain synchronization but would infer extra substitution errors. Substitutions in an agreed string thus provide a means of communicating information. Watermark codes exploit this idea. The message to be communicated is converted to a sparse string (the density of the sparse string determines the rate of the watermark code) which is then added modulo 2 to an agreed "watermark" string. The decoder receives the modified watermark string further corrupted by channel insertions, deletions, and substitutions, and attempts to identify the position of synchronization errors. Once this is done, the decoder can report a noisy version of the sparse string (to be precise, the decoder returns a likelihood function over the sparse strings).

As in the example above, the precise position of the synchronization errors cannot be determined exactly. We protect the sparse string with an outer code that allows us to recover the original message.

Before describing watermark codes in more detail, we introduce a model of an insertion/deletion channel with substitutions. We expect watermark codes to be useful for a large class of channels, subject to appropriate modifications of the decoding algorithm.

III. Channel Model

We consider a binary channel described by three parameters: P_{s}, P_{i}, and P_{d}, which control the rates of substitutions, insertions, and deletions [13]. We imagine the transmitted bits t_i entering a queue, waiting to be transmitted by the channel. At each channel use, one of three events occurs. With probability P_{i}, a random bit is inserted into the received stream. With

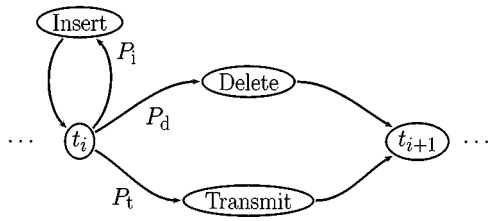

Fig. 1. Insertion/deletion channel with probabilities P_{i}, P_{d}, and P_{t} of insertions, deletions, and transmissions. The "Insert" state inserts a single random bit. The "Transmit" state makes a fraction P_{s} of substitution errors.

TABLE I
EXPLICIT EMISSION PROBABILITIES FOR INSERTION/DELETION CHANNEL MODEL WITH GEOMETRIC DISTRIBUTION OF INSERTED LENGTHS. $\alpha_I = 1/(1-(P_{\mathrm{i}})^I)$ IS A NORMALIZING CONSTANT TO TAKE ACCOUNT OF THE MAXIMUM INSERTED LENGTH I. EACH BIT t_i ENTERING THE CHANNEL RESULTS IN BETWEEN 0 AND I INSERTION EVENTS FOLLOWED BY EITHER THE TRANSMISSION OR DELETION OF t_i. A FRACTION P_{s} OF TRANSMISSION EVENTS SUFFER SUBSTITUTION ERROR

Insertions	Deletions	Substitutions	Probability
0	1	-	P_{d}
0	0	0	$P_{\mathrm{t}}(1-P_{\mathrm{s}})$
0	0	1	$P_{\mathrm{t}}P_{\mathrm{s}}$
1	1	-	$\alpha_I P_{\mathrm{i}} P_{\mathrm{d}}$
1	0	0	$\alpha_I P_{\mathrm{i}} P_{\mathrm{t}}(1-P_{\mathrm{s}})$
1	0	1	$\alpha_I P_{\mathrm{i}} P_{\mathrm{t}} P_{\mathrm{s}}$
⋮	⋮	⋮	
I	1	-	$\alpha_I (P_{\mathrm{i}})^I P_{\mathrm{d}}$
I	0	0	$\alpha_I (P_{\mathrm{i}})^I P_{\mathrm{t}}(1-P_{\mathrm{s}})$
I	0	1	$\alpha_I (P_{\mathrm{i}})^I P_{\mathrm{t}} P_{\mathrm{s}}$

probability P_{d}, the next queued bit is deleted. With probability $P_{\mathrm{t}} = (1 - P_{\mathrm{d}} - P_{\mathrm{i}})$, the next queued bit is transmitted, i.e., put into the received stream, with a probability P_{s} of suffering a substitution error. The channel is represented in Fig. 1. Under this model, the burst length of an insertion event has a geometric distribution. For computational convenience we will generally impose a maximum insertion length I.

In the discussion that follows we will assume that $P_{\mathrm{d}} = P_{\mathrm{i}}$, so that the expected length of the received string is equal to the transmitted length (except for a correction due to I of order P_{i}^I).

If we impose an upper limit I on the number of consecutive insertions then we can write out the channel model explicitly. Each input bit t_i gets mapped to between 0 and $I+1$ output bits, according to the probabilities in Table I.

IV. ENCODING

The concatenation of watermark codes with low-density parity-check codes is outlined in Fig. 2. Using a low-density parity-check code we first encode the message $\boldsymbol{m}$ of length K_L q-ary symbols into a vector $\boldsymbol{d}$ of length N_L. We use low-density parity-check codes [24], [25] defined over the field GF$(q = 2^k)$ [16] because they are good codes and can easily utilize the soft information provided by the inner code.

We define the *density* of a binary vector $\boldsymbol{x}$ as $h(\boldsymbol{x})/l(\boldsymbol{x})$, where $h(\boldsymbol{x})$ is the Hamming weight of $\boldsymbol{x}$ and $l(\boldsymbol{x})$ is the length of $\boldsymbol{x}$. A vector with density less than $1/2$ is said to be *sparse*.

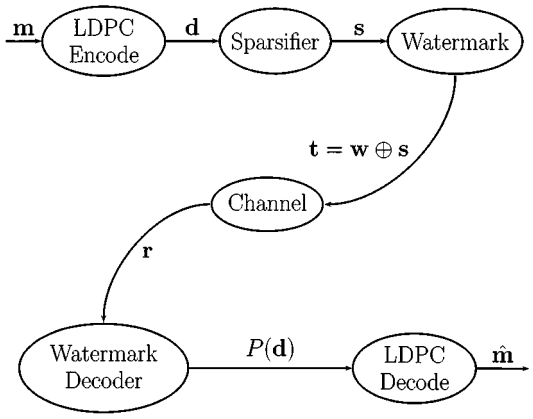

Fig. 2. Overview of watermark code construction.

The low-density parity-check codeword $\boldsymbol{d}$ is transformed into a sparse binary vector $\boldsymbol{s}$: we choose some $n > k$ and map each q-ary symbol of $\boldsymbol{d}$ into one of the 2^k lowest density binary vectors of length n using a lookup table. We denote the mean density of the sparse vectors by f. The overall block length of the code is $N := nN_L$ and the rate of the code is $\frac{K_L}{N_L} \times \frac{k}{n}$.

The sparse vector $\boldsymbol{s}$ is added, modulo 2, to the watermark string $\boldsymbol{w}$ (which is known to both encoder and decoder) to produce the transmitted vector $\boldsymbol{t}$. The watermark string provides the synchronization capability of the code so it is important that local translations of the watermark string are easily identified. Useful choices of $\boldsymbol{w}$ include random and run-length-limited sequences.

The watermark (inner) decoder processes the noisy received vector $\boldsymbol{r}$ and produces a set of N_L likelihood functions, one for each symbol of the low-density parity-check codeword $\boldsymbol{d}$. These likelihoods initialize the outer decoding algorithm.

V. DECODING

We model the received vector as having been produced by a hidden Markov model (HMM): i.e., we ignore the correlations between substitutions that are caused by the construction of $\boldsymbol{s}$ and the structure in the outer low-density parity-check codeword. These correlations could be taken into account, at an increased computational cost. Given these simplifying assumptions, the optimal solution to the synchronization problem uses dynamic programming (the forward–backward algorithm), as we now describe.

We define the synchronization *drift* x_i at position i to be the (number of insertions) $-$ (number of deletions) made by the channel from the first transmitted bit t_0 to the point where bit t_i is ready to be transmitted. Explicitly, if bit t_{i-1} is not deleted then it appears in the received stream as r_{i-1+x_i}. The sequence $\{x_i\}$ will form the hidden states of an HMM [26]. We expect x_i to perform a random walk; if P_{i} and P_{d} are equal then this walk has mean zero and variance that grows linearly with i.

The synchronization task can be conveniently represented on a lattice. In Fig. 3, the received vector is placed along the top and the transmitted vector is placed down the side. In this representation, deletions are indicated by vertical lines, insertions by

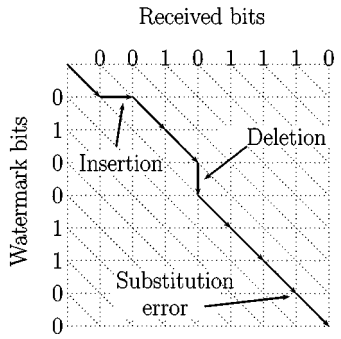

Fig. 3. Lattice representation of synchronization. The solid line shows one possible alignment of the received bits with the watermark bits. There is an insertion (horizontal line) after the first bit, a deletion of the fourth bit, and a substitution error at the seventh bit.

horizontals, and transmissions by diagonals. A path that passes through the point (i, j) associates received bit r_j with watermark bit w_i (or with a bit inserted between w_i and w_{i+1}). A diagonal (transmission) step from (i_1, j_1) to (i_2, j_2) defines the drift $x_{i_2} = (j_2 - i_2)$.

A. Hidden Markov Model (HMM)

The model is parameterized by the channel parameters P_{i}, P_{d}, P_{s}; the mean density of the sparse vectors f; and by the choice of watermark vector $\boldsymbol{w}$. We assume that the sparse bits of $\boldsymbol{s}$ are independent and identically distributed. It will be useful to denote the effective substitution rate, the probability that a bit transmitted by the channel is not equal to the corresponding watermark bit, by

$$P_{\mathrm{f}} = f(1 - P_{\mathrm{s}}) + (1 - f)P_{\mathrm{s}}. \tag{1}$$

The parameters will collectively be denoted by $\mathcal{H}$. We use the channel model outlined in Section III.

States x_i in the model take values in the alphabet

$$\boldsymbol{X} := \{\ldots, -2, -1, 0, 1, 2, \ldots\}.$$

Each transmitted bit produces from 0 to $I+1$ received bits. Hence, if $x_i = a$, then x_{i+1} must take values in $\{a-1, \ldots, a+I\}$, representing the range from a deletion event $(x_{i+1} - x_i = -1)$ to an insertion of I bits followed by transmission of $t_i (x_{i+1} - x_i = I)$.

The transition matrix entry P_{ab} contains the prior probability $P(x_{i+1} = b | x_i = a)$. Such a transition can occur in two ways: either the channel makes $(b - a + 1)$ insertions and deletes transmitted bit t_i, or the channel makes $(b - a)$ insertions and transmits t_i. In general, there are two contributions to P_{ab}. In both cases, $(b - a + 1)$ bits are emitted by the channel. Thus

$$P_{ab} = \begin{cases} P_{\mathrm{d}}: & b = a - 1 \\ \alpha_I P_{\mathrm{i}} P_{\mathrm{d}} + P_{\mathrm{t}}: & b = a \\ \alpha_I \left((P_{\mathrm{i}})^{b-a+1} P_{\mathrm{d}} + (P_{\mathrm{i}})^{b-a} P_{\mathrm{t}} \right): & a < b < a + I \\ \alpha_I (P_{\mathrm{i}})^I P_{\mathrm{t}}: & b = a + I \\ 0: & \text{otherwise} \end{cases} \tag{2}$$

where $\alpha_I = 1/(1 - (P_{\mathrm{i}})^I)$.

The transition from $(x_i = a)$ to $(x_{i+1} = b)$ results in the emission of $(b - a + 1)$ bits. We can define α and β by $P_{ab} = \alpha P_{\mathrm{d}} + \beta P_{\mathrm{t}}$, where α is the probability of making $(b - a + 1)$ insertions, and β is the probability of making $(b - a)$ insertions. We can write $Q^i_{ab}(s)$, the conditional probability of emitting the string s given the transition $(x_i = a)$ to $(x_{i+1} = b)$ (note that the Q^i values depend on the watermark bit w_i) as follows:

$$Q^i_{ab}(s) = \begin{cases} \frac{\alpha P_{\mathrm{d}} / 2^{b-a+1} + \beta P_{\mathrm{t}} (1 - P_{\mathrm{f}}) / 2^{b-a}}{P_{ab}}: & s^* = w_i \\ \frac{\alpha P_{\mathrm{d}} / 2^{b-a+1} + \beta P_{\mathrm{t}} P_{\mathrm{f}} / 2^{b-a}}{P_{ab}}: & s^* = w_i \oplus 1 \end{cases} \tag{3}$$

where $s^* := s_{b-a+1}$ is the received bit associated with watermark bit w_i. Then $\sum_s Q^i_{ab}(s) = 1$, where the summation is over all binary strings s of length $(b - a + 1)$.

B. Inner Decoding

The objective of the inner decoding algorithm is to output a likelihood function over vectors $\boldsymbol{d}$, $P(\boldsymbol{r}|\boldsymbol{d}, \mathcal{H})$, which can be used by the outer low-density parity-check decoder. We ignore the correlations in the likelihood function and calculate the symbol-by-symbol likelihood $P(\boldsymbol{r}|d_i, \mathcal{H})$ via a forward–backward algorithm using the HMM described in the previous section. Define the forward quantity

$$F_j(y) = P(r_1, \ldots, r_{j-1+y}, x_j = y | \mathcal{H})$$

the probability that the drift at position j is y and that the first $(j - 1 + y)$ bits emitted by the channel agree with $\boldsymbol{r}$. Similarly, the backward quantity

$$B_j(y) = P(r_{j+y}, \ldots | x_j = y, \mathcal{H})$$

denotes the probability of emitting the tail of $\boldsymbol{r}$ given a drift of y at position j. Then

$$P(\boldsymbol{r}|d_i, \mathcal{H}) = \sum_{x_{i_-}, x_{i_+}} F_{i_-}(x_{i_-})\, P(\boldsymbol{r}^0, x_{i_+} | x_{i_-}, d_i, \mathcal{H})\, B_{i_+}(x_{i_+}) \tag{4}$$

where $i_- = n \times i$ and $i_+ = n \times (i+1)$; i.e., d_i is encoded into the n sparse bits $(s_{i_-}, \ldots, s_{i_+ - 1})$. Here $\boldsymbol{r}^0$ denotes the received bits $(r_{(i_- + x_{i_-})}, \ldots, r_{(i_+ + x_{i_+} - 1)})$.

Because a transmitted bit results in an unknown number of received bits, we use an output probability distribution that depends on the *transition* from x_{i-1} to x_i rather than on the state x_i. The output distributions Q are given in (3). Thus, the forward quantities are calculated as follows:

$$F_j(y) = \sum_{a=y-I}^{y+1} F_{j-1}(a) P_{ay} Q^{j-1}_{ay}(r_{j-1+a}, \ldots, r_{j+y-1}) \tag{5}$$

Calculation of the backward quantities is done similarly.

The bits of the sparse vector $\boldsymbol{s}$ are not independent: the sparse bits $(s_{i_-}, \ldots, s_{i_+ - 1})$ are determined by the value of d_i. For each possible value of d_i we calculate the likelihood $P(\boldsymbol{r}^0, x_{i_+} | x_{i_-}, d_i, \mathcal{H})$ by fixing $(s_{i_-}, \ldots, s_{i_+ - 1})$ according to d_i and performing a forward pass between x_{i_-} and x_{i_+}. In this pass, $w_i \oplus s_i$ is known, so all substitution errors are due to

the channel. Hence the emission probabilities are as in (3) with w_i replaced with $w_i \oplus s_i$ and P_f with P_s.

To reduce complexity, we limit the maximum allowed drift to $|x_i| \leq x_{\max}$, although this condition can be relaxed (see Section VII-A). We choose $x_{\max}$ to be several times larger than the standard deviation of the synchronization drift over one block length, given by $\sqrt{NP_\mathrm{d}/(1-P_\mathrm{d})}$ [18]. The decoding complexity is proportional to $Ix_{\max}$.

In principle, the forward/backward passes can be performed over all of the received data at once. To reduce decoding delay, we decode the inner code one block at a time using a sliding window on the received data. The decoder infers the position in the received stream of block boundaries (as described in the next subsection) and slides the decoding window accordingly. The window is anchored at the most likely start-of-block position. As the channel gets noisier, synchronization becomes more difficult and the error in the identification of the block boundaries increases. If the accumulated errors exceed $x_{\max}$ then the decoder cannot resynchronize the received stream and decoding fails. We protect against such catastrophes by providing a mechanism for detecting and correcting gross loss of timing, as discussed in Section V-E.

When performing sliding-window decoding, it is computationally convenient to run the forward pass several (e.g., five) multiples of $x_{\max}$ beyond the expected position of the block boundary and to initialize the backward pass from the final forward values.

C. Sliding-Window Decoding

If the decoder is not told the position of the block boundaries, that position can be inferred. The most likely value of the drift at the end of the block, $\hat{x}_{N+1} = \arg\max_y F_{N+1}(y)B_{N+1}(y)$, is used to slide the decoding window to the start of the following block.

We move the zero point of our received stream by $(N+\hat{x}_{N+1})$ positions. The quantities F are shifted accordingly

$$F_1(y) \leftarrow \alpha \times \begin{cases} F_{N+1}(y+\hat{x}_{N+1}): & |y+\hat{x}_{N+1}| \leq x_{\max} \\ 0: & \text{otherwise} \end{cases} \tag{6}$$

where α is a normalizing constant such that $\sum_y F_1(y) = 1$. The decoder can then proceed to synchronize the next block.

D. Outer Decoding

The likelihoods (4) are used to initialize the low-density parity-check decoder. The uncertainty associated with the likelihood is time-varying, with greatest uncertainty in the vicinity of insertion and deletion events. The sum–product algorithm for the low-density parity-check decoding makes appropriate use of this soft information.

E. Acquiring Synchronization

When decoding watermark codes, it may be necessary to recover lost synchronization or to start decoding a stream with unknown synchronization. For example, if consecutive blocks fail to decode we should suspect global loss of synchronization. The method used to recover synchronization is the same as that described in Section V-C for finding the end of a block during normal decoding; the difference lies in the initial conditions for the forward/backward passes. In practice, synchronization can be maintained well beyond the noise level at which the outer decoder fails.

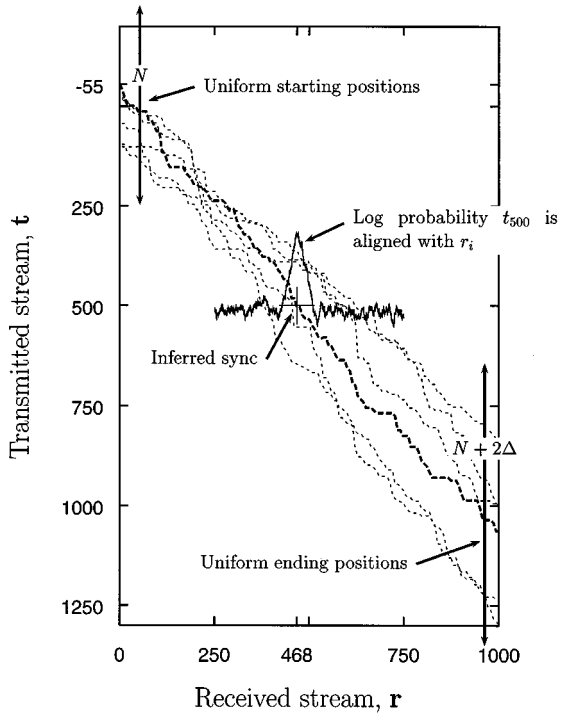

Fig. 4. Resynchronization (schematic diagram). We initialize the forward and backward passes as given in (7) and (8). Using two blocks of received data, the forward–backward algorithm is used to calculate the posterior probability of the position in the received stream of the start of the next block. In this example, $N = 500$, $\Delta = 100$, and the assumed zero point is close to the true start of block. The true path is emphasized. The other dotted lines represent samples from the prior. The position of the end of the block is correctly identified as position 468 in the received stream.

To find the block boundary, we proceed as follows. We choose some arbitrary position in the received stream to have index 0. We assign uniform priors to the transmitted block positions of the zeroth and $2N$th received bits by initializing the forward and backward passes as follows:

$$F_{-n}(n) = \begin{cases} 1/N, & \text{if } -N/2 \leq n < N/2 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

$$B_{2N-n}(n) = \begin{cases} 1, & \text{if } 2N - N/2 - \Delta \leq n < 2N + N/2 + \Delta \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

where $\Delta = 2x_{\max}$ (say). We perform forward–backward decoding from position 0 to $2N$ in the received stream, to infer the most likely value of x_N:

$$P(x_N = y|\boldsymbol{r}) \propto F_N(y)B_N(y). \tag{9}$$

The method is shown graphically in Fig. 4.

When synchronization is lost, the decoder can usually restrict attention to a narrow range of possible synchronizations cen-

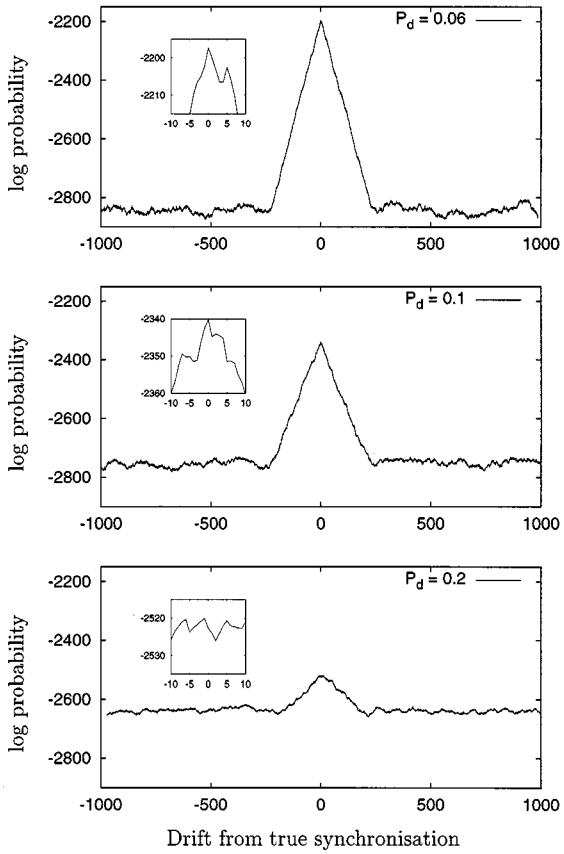

Fig. 5. Using a uniform prior over possible synchronization points (i.e., start of block), the forward–backward algorithm is used to calculate a posterior distribution. Horizontal axis: distance from inferred start of block to true start of block. Vertical axis: log (base e) probability (not normalized) of the inferred start of block. Results are shown for several choices of P_{d}, increasing from top to bottom. The correct synchronization is easily determined (with a small margin of error), even when the probability of insertion and deletion is 20%. $P_{\mathrm{f}} = 0.125$, block length 2000 bits.

tered on the last known value, rather than using the broad priors described above.

Fig. 5 shows the logarithm of the unnormalized posterior probability $F_N(y)B_N(y)$ as a function of the distance from the inferred start of the next block (i.e., $N + y$) to the true position. The results are for a watermark code of length 2000 bits with a sparse vector $\boldsymbol{s}$ of density 0.125, and a channel with $P_{\mathrm{s}} = 0$. Using such a watermark code concatenated with a rate-$1/2$ inner code over GF(8), we can communicate reliably if $P_{\mathrm{d}} < 0.05$. The correct synchronization is still easily identifiable for $P_{\mathrm{d}} = 0.1$ (the noise level at which decoding becomes unreliable for inner codes of rate $1/10$). Even for $P_{\mathrm{d}} = 0.2$, although there is usually a small error in the identification of the block boundary, the errors in the inferred drift are bounded to within $\approx$20 by the exponential drop in posterior probability away from the true block boundary.

It is also possible to regain synchronization using a Viterbi algorithm along a narrow corridor in the lattice. If the corridor does not contain the true path, then the probability of the Viterbi path will be similar to that for a random transmitted stream. On the other hand, if the corridor does contain the true path, then the Viterbi path will have larger probability, which is easily detectable. Such methods might be useful for a rough estimate of the synchronization, but we do not expect the estimates of the Viterbi algorithm to be as useful as the sum–product approach. Viterbi paths are often not "typical" paths, in that the number of insertions, deletions, and transitions may vary significantly from the expected and true counts. The sum–product algorithm is the correct algorithm for inferring the synchronization. Nevertheless, at lower noise levels the Viterbi method is generally good enough for resynchronization and is a cheaper alternative than the sum–product algorithm.

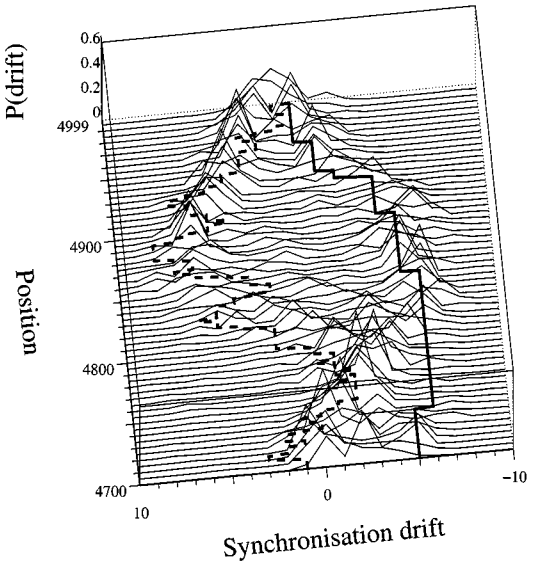

Fig. 6. Comparison of Viterbi path (solid) and true path (dashed) to the pointwise densities calculated by the sum–product algorithm for a short section of a block. Insertion/deletion probabilities: $P_{\mathrm{d}} = P_{\mathrm{i}} = 0.1$; substitution probability: $P_{\mathrm{f}} = 0.35$.

The noticeable differences between the pointwise synchronization estimates obtained with the sum–product algorithm and the corresponding Viterbi path are illustrated graphically in Fig. 6. The probability density generated by the sum–product decoder is compared with the Viterbi path and the true path. Notice how much straighter the Viterbi path is than the true path.

VI. Experiments and Results

A. Simulation Method

The performance of watermark codes was tested by simulation as follows. A low-density parity-check code of length N_L over GF$(q = 2^k)$ and a sparseness parameter $n > k$ were chosen. A binary pseudorandom watermark vector $\boldsymbol{w}$ of length $N_L \times n$ was created.

For convenience, we avoided explicitly generating low-density parity-check codewords when simulating watermark codes by using the following procedure. A long random string $\boldsymbol{d}$ of q-ary symbols was generated. This string was translated via the lookup table to a sparse string $\boldsymbol{s}$ and added to the watermark vector (which repeated every nN_L bits) to produce the transmitted string. The watermark decoder returned a distribution for

TABLE II
PARAMETERS OF CONCATENATED WATERMARK CODES REPORTED IN THIS PAPER. FOR EACH CODE WE REPORT: TRANSMITTED BLOCK LENGTH IN BITS, N; OVERALL RATE, R; TRANSMITTED BLOCK LENGTH OF OUTER CODE, N_L; SOURCE BLOCK LENGTH OF OUTER CODE, K_L; BITS PER SYMBOL OF OUTER CODE, k (OUTER CODE IS DEFINED OVER GF(2^k) AND HAS TRANSMITTED BLOCK LENGTH $k \times N_L$ BITS); SPARSENESS PARAMETER n; RATE OF WATERMARK CODE $r_w := k/n$

Code	N	R	N_L	K_L	k	n	r_w
A	2500	0.40	500	250	4	5	0.80
B	3000	0.33	500	250	4	6	0.67
C	4662	0.21	666	333	3	7	0.43
D	4995	0.71	999	888	4	5	0.80
E	4000	0.50	800	500	4	5	0.80
F	4002	0.50	667	500	4	6	0.67
G	4662	0.21	777	333	3	6	0.50
H	4662	0.21	666	333	3	7	0.43
I	6000	0.05	1000	100	3	6	0.50

each q-ary symbol. Let $\hat{d}_i$: GF$(q) \rightarrow \mathbb{R}$ be the distribution returned for symbol d_i. The ith noise symbol for the low-density parity-check code was defined to be

$$n_i = d_i - \arg\max_a(\hat{d}_i(a))$$

and the distribution for the ith noise symbol was then

$$\hat{n}_i(a) = \hat{d}_i(a + d_i - n_i)$$

(arithmetic in GF(q)). The low-density parity-check decoder then performed syndrome decoding using the distributions $\{\hat{n}_i\}$ for each symbol i.

The drift was set to zero at the start. A continuous stream of received bits was supplied to the decoder and for subsequent blocks the decoder was fully responsible for retaining synchronization. Typical cumulative synchronization error at the end of every block was $\pm$ 1 bit, even after many thousands of blocks had been decoded. For decoding to be possible, synchronization errors must remain bounded. Very occasionally, synchronization errors accumulated and several blocks in a row were incorrectly decoded. In all cases, increasing $x_{\max}$ and performing a second sum–product decoding (see Section V-E) successfully regained the lost synchronization.

For high-rate low-density parity-check codes (e.g., rate 0.7) synchronization was never lost. In this regime, fewer errors can be corrected by the low-density parity-check decoder and hence, for channels over which communication is possible, synchronization is very well determined. Resynchronization errors made by the watermark decoder are typically caused by the displacement of insertion/deletion events by one or two positions.

Table II shows the parameters of all codes whose performance is reported in this paper.

B. High-Rate Watermark Codes

Fig. 7 shows the decoding accuracy of code D, an $N = 4995$, $K = 3552$ concatenated watermark code, as a function of the channel parameters. The code was constructed from a rate–$8/9$ low-density parity-check code over GF(16) [17]. Each symbol was mapped to 5 bits of $\boldsymbol{s}$, which made the density of the sparse vector 0.3125. A block error rate of less than 10^{-3} was achieved for insertion/deletion probability 1.5×10^{-3} and channel substitution error rate less than 3×10^{-3}. At these noise levels there are an average of 14 synchronization and 14 substitution errors per block.

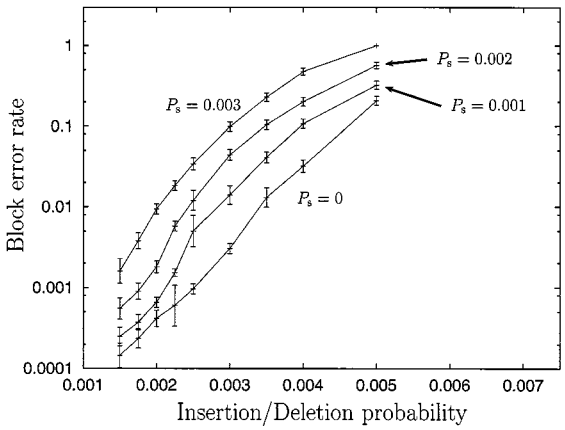

Fig. 7. Block error rate as a function of insertion/deletion probabilities $P_d = P_i$, for several choices of substitution probability P_s. Inner code: Watermark code with $n = 5$. Outer code: Regular low-density parity-check code over GF(16) of column weight 3, rate $8/9$, length 999. Overall rate 0.71, with a total block length of 4995 bits.

Results are shown for several settings of the channel substitution rate P_s. Increasing P_s affects the performance smoothly, as suggested by the entropy surface (Fig. 14). For the common case of channels with substitution rates much less than the density of $\boldsymbol{s}$, synchronization performance is not significantly affected. The effect of channel substitutions is to increase the entropy of the input to the low-density parity-check decoder. As a result, the degradation in performance as P_s increases is comparable to that for a binary-symmetric channel with increasing noise, with the insertion/deletion errors adding an effective background noise level. Subsequent results are quoted for channels without substitution errors.

C. Lower Rate Watermark Codes

For lower rate watermark codes, much higher rates of insertions, deletions, and substitutions can be tolerated. By reducing the rate of the watermark code, we can reduce the density of $\boldsymbol{s}$, making it easier to identify insertion/deletion errors. For example, with an outer code over GF(16) the density of $\boldsymbol{s}$ can be almost halved (from 0.31 to 0.17) by reducing the rate of the watermark code from 0.8 to 0.5. In Fig. 8, the performances of various watermark codes are compared. Results are shown for codes of rates approximately $0.7, 0.5, 0.2$, and 0.05. For clarity, all results are for channels without substitution errors.

For the results shown, the length of the low-density parity-check code was no more than 1000 q-ary symbols. For longer codes, we expect to see further improvements in performance. Improvements could also be made by using irregular constructions for the low-density parity-check codes [18], [27]. Nevertheless, the results show that watermark codes are extremely effective at communicating over channels with insertions and deletions.

There are few results in the literature that are directly comparable to those presented here. Levenshtein's explicit construc-

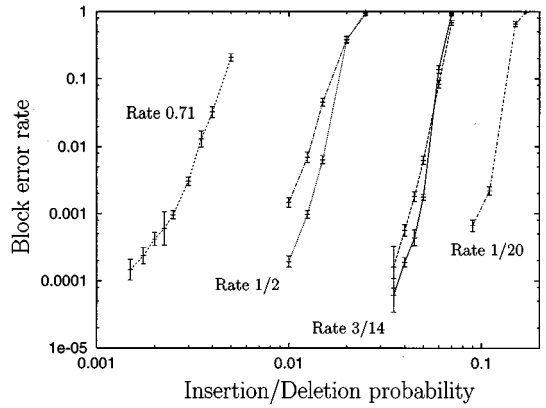

Fig. 8. Performance of concatenated watermark codes. The full parameters of the codes are given in Table II. Codes from left to right: Code D, rate 0.71; Codes E and F, rate $1/2$; Codes G and H, rate $3/14$; Code I, rate $1/20$. All outer codes were regular low-density parity-check codes over GF (2^k) with mean column weights between 2.6 and 3. The channel substitution probability P_s was zero.

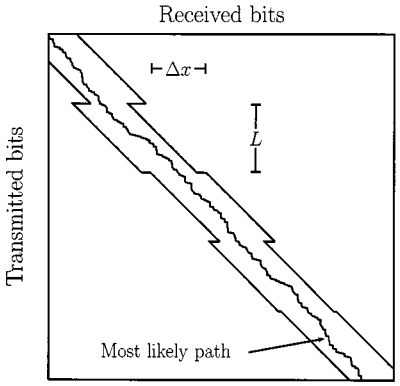

Fig. 9. Use of local synchronization to restrict the decoding lattice: Every L bits, the local synchronization is estimated using a short backward pass. This is used to define a "corridor" in the lattice of width Δx around the most likely path. Only paths within the corridor are considered during decoding. The expected drift in length L is a factor $\sqrt{L/N}$ smaller than for the whole block, so reducing the number of states required in the HMM decoder.

tions [7] could only correct single insertion or deletion errors. Most subsequent constructions correct a strictly limited number of synchronization errors (e.g., less than three), or else have no known practical decoding algorithm. Recently, Bours [28, Sec. 3.5.2] developed codes that could correct bursts of insertions and deletions. Rate 0.465 codes of block length 15 840 bits were presented which had a block-error rate 10^{-3} given a channel that made insertion/deletion bursts of expected length 6 on average once every nine blocks. Watermark codes can do much better, even if the synchronization errors are uncorrelated (which makes them have higher entropy than bursty errors). Fig. 8 shows watermark codes of rate 0.5 reaching block error rate 10^{-3} with roughly 100 synchronization errors scattered throughout each block of length 4002 bits.

VII. Complexity and Implementation Considerations

A. Faster Decoding

A naïve implementation of watermark decoding would result in a costly decoding algorithm. The cost of the forward–backward algorithm scales as XNI, where X is the number of states in the HMM $(X = 1+2x_{\max})$, N is the length of the hidden sequence, and I is the maximum length of a burst of insertions. For watermark codes, X limits the loss in synchronization that we consider within a block. Assuming that the insertion and deletion probabilities are equal, the synchronization drift in a block of length N is a Gaussian random variable with mean 0 and variance $N \times P_d/(1-P_d)$ [18]. If $P_d = 0.05$ and $X = 70$, then for blocks of length 4000 1% of blocks will overflow; we can deal with such blocks by increasing X and repeating the decoding.

An $O(XNI)$ algorithm is wasteful for two reasons. First, the synchronization is generally very well determined: for all codes and noise levels shown in Fig. 8 the average synchronization error at the end of a block was less than 1. Second, most of the entries in the HMM transition matrix represent weak or disallowed transitions, because the synchronization only drifts a small amount at each step.

These observations allow great reduction in the complexity of the decoding algorithm, at a slight expense of accuracy. Given our simple channel model, a burst of I insertions followed by a transmission is just as likely as two bursts of $I/2$ insertions separated by a transmission. In practice, in the region of such an event it would be impossible to distinguish transmitted from inserted bits. Hence, when decoding, it is sufficient to consider only synchronization drifts from -1 to $+2$ at each step (i.e., set $I = 2$). Longer insertion events will then be "explained" by several shorter events, at minimal cost to decoding accuracy.

Also, by performing short backward passes as the forward pass progresses, the local synchronization within a block can be estimated to within some uncertainty Δx. Then it is necessary only to propagate paths starting within the identified region. Even without short backward passes, useful speedups can be achieved by considering only forward/backward pass states with probabilities above a certain threshold. Combining these changes, the complexity of the decoding algorithm scales as $4\Delta xN$ rather than XNI.

Furthermore, by identifying the local synchronization throughout the block, we obviate the need for the artificial limit X on the synchronization drift. With appropriate modifications of the decoding algorithm, it is possible to perform the forward and backward passes through a corridor of width Δx surrounding the most likely path through the lattice (Fig. 9).

B. Choice of Watermark Vector

It is possible that watermark vectors with structure could have useful properties not offered by purely random choices. Long runs of zeros or ones make it impossible to localize deletions (and 50% of insertions) to greater accuracy than the length of the run. Hence, it is natural to consider using run-length limited (RLL) sequences as watermark vectors. (Constraining the minimum run-length to be greater than 1 is not expected to be useful.)

RLL watermark sequences with various maximum run-lengths have been compared. The sequences were constructed using a uniform random bit generator whenever the next bit

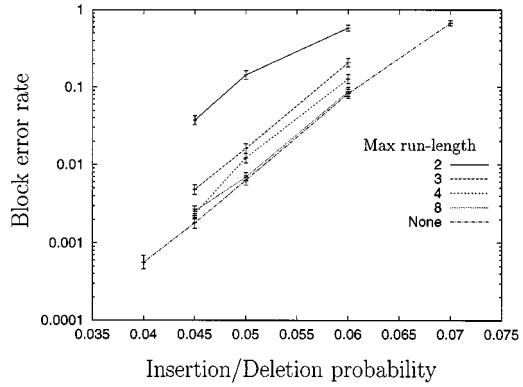

Fig. 10. Results for codes with maximum RLL watermark vectors. Overall rate of code was $3/14$, block length 4662 bits (code C). Outer code: regular rate $1/2$ low-density parity-check code over GF(8).

in the sequence was not determined by the RLL constraints. While it is likely that the worst case performance of codes with unconstrained random watermark sequences is worse than that of codes with sequences satisfying moderate RLL constraints, no choice of RLL constraints has been found to improve average performance. As shown in Fig. 10, watermark sequences with maximum run-length less than 4 gave significantly poorer performance. As the maximum run-length was increased, the results soon became indistinguishable from those obtained for random watermark sequences.

We have also tried using a Thue–Morse nonrepeating sequence [29] as the watermark sequence, but we have not been able to find any sequence that gives better average performance than the random watermark.

VIII. Future Directions

A. Channel Models

It would be straightforward to develop watermark codes and decoders for different channels. For example, many channels of practical interest (e.g., the S-DAT recording channel [30]) exhibit occasional bursts of synchronization errors as well as more frequent single or double errors. If we change the HMM used by the watermark decoder, we can use watermark codes to communicate over such channels without further modifications.

B. Irregular Watermark Codes

In marker codes [6], information-carrying sections of a block are interspersed with synchronization-providing marker sections. Marker codes can be viewed as a special case of "irregular" watermark codes, i.e., watermark codes in which the density of the sparse vector $\boldsymbol{s}$ is not uniform. The marker sections correspond to regions where $\boldsymbol{s}$ has a density of 0, and information sections to regions where $\boldsymbol{s}$ has a density 0.5 and the watermark is set to zero.

By varying the density of $\boldsymbol{s}$ the information can be spread unevenly throughout the block with some sections being used to provide easier synchronization. Information about variations in the density of $\boldsymbol{s}$ is easily incorporated into the decoding algorithm. Preliminary work by Ratzer [31] suggests that marker codes, decoded using the forward–backward algorithm, may perform better than regular watermark codes.

If irregular low-density parity-check codes are used as the outer code, a possible choice is to vary the density of $\boldsymbol{s}$ according to the weight of the corresponding column of the parity-check matrix. We conjecture that a good construction would connect the high weight columns of the parity-check matrix to the sparser sections of $\boldsymbol{s}$.

C. Iterative Watermark Decoding

It is possible to use an iterative decoding scheme for the watermark code. When the watermark decoder first runs, all it knows about $\boldsymbol{s}$ is the expected density. After one or more steps of low-density parity-check decoding, the current estimate of $\boldsymbol{d}$ could be used to calculate a refined estimate of the bits of $\boldsymbol{s}$. This could be used by the watermark decoder to generate a new estimate of $\boldsymbol{d}$. In this way, information passes between the outer and inner decoders using the sum–product algorithm. It is likely that significant improvements could be obtained with such an algorithm.

Appendix
Limits of Watermark Codes

Little is known about the capacity of the insertion/deletion channel under consideration, beyond the bounds of Ullman and Zigangirov [12], [13].

In this appendix, we investigate the resynchronization and decoding limits of watermark codes. We start by analyzing the ability of watermark codes to identify the position of insertion/deletion errors in the received stream. We show that catastrophic loss of synchronization is unlikely whenever insertion/deletion rates are low enough to admit decoding with block-error rates below 10^{-1}. Second, we examine the quality of the information passed to the outer decoder. This allows us to conjecture lower bounds on the capacity of specific insertion/deletion channels.

A. Limits of Synchronization

In this section, we measure the ability of watermark codes to resynchronize the received stream. One measure of this ability is the fraction of positions in which the decoder correctly determines the synchronization drift (i.e., in which the estimated drift $\hat{x}_i := \arg\max_y F_i(y)B_i(y)$ is equal to the true drift x_i). By examining how this fraction depends on the channel parameters, we can estimate the maximum density of the sparse vector $\boldsymbol{s}$ for which reliable synchronization is possible. The fraction required for successful outer decoding depends on the outer code rate, but is generally at least 0.5.

In identifying insertion/deletion errors, the watermark decoder makes no distinction between watermark vector substitutions that are due to the channel and those that are due to $\boldsymbol{s}$. Hence, for given P_{i} and P_{d}, the synchronizability of the received stream depends only on the effective substitution rate P_{f}.

Consider the model sketched in Fig. 11 using the lattice notation. This is equivalent to the channel of Fig. 1 with no maximum insertion length. Transmissions suffer a fraction P_{f}

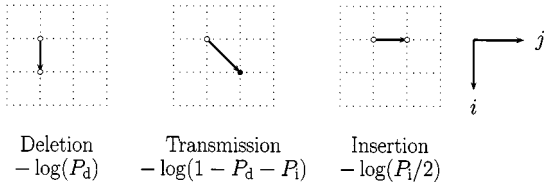

Fig. 11. A simple insertion/deletion channel. Received bits run along the horizontal direction, watermark bits along the vertical. In this model, each transition consists of a single deletion, insertion, or transmission. The transitions are independent and occur with fixed probabilities, i.e., any combination of the three transition types may be concatenated to produce a valid path. Costs of the transitions are given by the log-probabilities shown. There is a further cost for transmissions (solid circle) arriving at a node (i, j), which is $-\log(1-P_f)$ (or $-\log(P_f)$) if the received bit r_j matches (or does not match) the watermark bit w_i.

of substitutions. This representation lends itself to an efficient implementation of a Viterbi decoder in integer arithmetic, making it practical to run experiments over moderately long block lengths with less restrictive constraints on the maximum synchronization drift. As usual, we take $P_d = P_i$.

To estimate the range of channel parameters for which synchronization errors remain bounded, the following method was used. Blocks of 30 000 random watermark bits were created. These were passed through the simulated channel to produce received bits. A Viterbi decoder was used to find the most probable path through the lattice. The decoder was initialized with the true synchronization (at the top left of the lattice) but was not given the end point. The maximum allowed drift (200) was between two and five times larger than the typical drift in an unconstrained model.

The fraction of correctly determined drifts (the "fidelity"), averaged over multiple blocks, was recorded. In Fig. 12, the fidelity is plotted as a function of the insertion/deletion and substitution probabilities. When the insertion/deletion probability is low, good resynchronization is possible even for quite high substitution levels. For example, with $P_d = 0.04$ and $P_f < 0.3$, over half the received bits were correctly resynchronized. We consider such high effective substitution probabilities P_f because high-rate codes require relatively dense vectors $\boldsymbol{s}$. For example, with a low-density parity-check code over GF(16), to construct a rate–$4/5$ watermark code requires a sparse vector of density 0.3125. That is, even if the channel makes no substitutions the effective substitution probability P_f is over 30%. In our experience, with outer codes of rate greater than $1/2$ we require a fidelity of at least 0.8 to obtain a block-error rate of less then 10^{-3}. For rate–$1/10$ outer codes, a fidelity of 0.5 is sufficient.

B. Limits of Decoding

Synchronization is only half the battle. To make a decoding, the low-density parity-check decoder needs a reasonable estimate of the q-ary symbols of the low-density parity-check codeword $\boldsymbol{d}$. In this section, the factors affecting the quality of this estimate are investigated. Lower bounds on the capacity of insertion/deletion channels are conjectured.

The dominant factors affecting the quality of the estimates passed to the low-density parity-check decoder are the density f of the sparse vector $\boldsymbol{s}$, and the order of the field GF(q). The

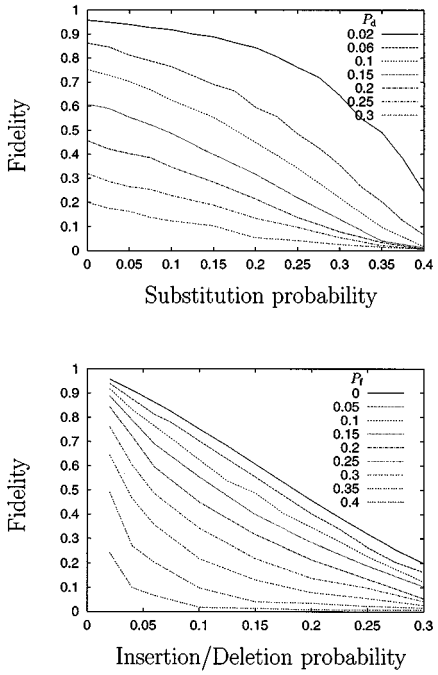

Fig. 12. Top: fraction of correctly resynchronized bits (fidelity) as a function of substitution probability, for several settings of P_d, with $P_i = P_d$. Bottom: fraction of correctly resynchronized bits as a function of insertion/deletion probability, for several settings of P_f.

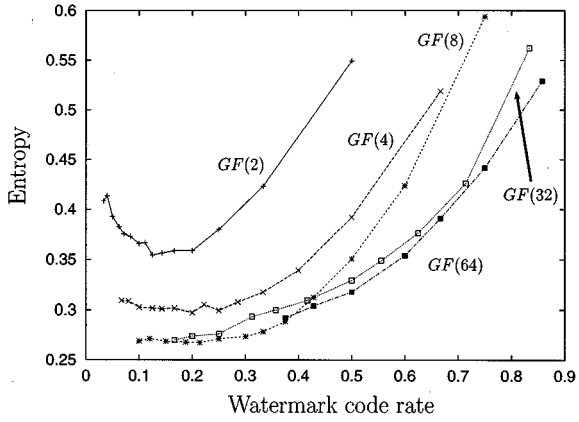

Fig. 13. Average entropy of the symbol-by-symbol likelihoods $P(\boldsymbol{r}|d_i)$ returned by the watermark decoder, as a function of the watermark code rate. Channel parameters: $P_d = P_i = 0.05$, $P_s = 0$.

density f is a function of q and n, the number of sparse bits used to represent a q-ary symbol. f limits the accuracy with which the watermark decoder determines the synchronization, while $\log(q)/n$ is the rate of the watermark code. For a given density f we would like to make the rate $\log(q)/n$ as large as possible.

Fig. 13 shows the average conditional entropy per bit of $\boldsymbol{d}$ given $\boldsymbol{r}$ as a function of the watermark code rate $\log(q)/n$. To be precise, we show what the conditional entropy would be in the absence of the outer code, namely, the average entropy of the normalized likelihoods $P(\boldsymbol{r}|d_i)$ calculated by the watermark decoder (see (4)). We now discuss the reasons for the observed dependencies shown in the figure.

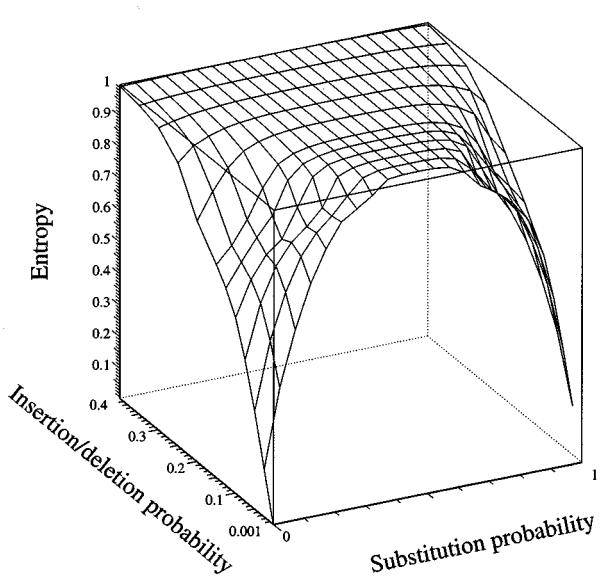

Fig. 14. Conditional entropy $H(\boldsymbol{d}|\boldsymbol{r})$ as a function of the channel parameters $P_{\mathrm{d}} = P_{\mathrm{i}}$, P_{s}. Outer low-density parity-check code defined over GF (8), block length 666. Here q-ary symbols are mapped to groups of seven sparse bits (code C in Table II). The per-symbol entropy of the likelihoods $P(\boldsymbol{r}|d_i)$ has been divided by $\log(q)$. Results have been averaged over 50 blocks.

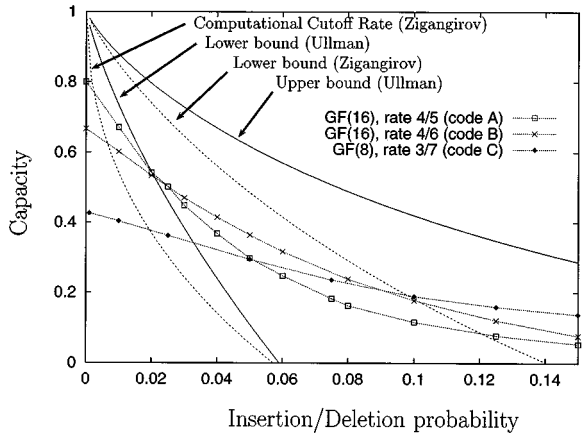

Fig. 15. Bounds on the capacity of an insertion/deletion channel with no substitutions. Solid lines show upper and lower bounds calculated by Ullman; dotted lines show the lower bound of Zigangirov, and the computational cutoff rate for tree-like convolutional codes. The points show empirical conjectured lower bounds obtained using watermark codes of rate $3/7$, $4/6$, and rate $4/5$. The parameters of these three codes are given in Table II.

For a fixed watermark code rate $\log(q)/n$, the density f decreases (albeit nonmonotonically) as $\log(q)$ increases. For increasing q and n, $\log(q)/n \to H_2(f)$. Hence, for a given watermark code rate, we can make synchronization easier simply by increasing q (and n) appropriately. The utility of this approach is limited by the $O(q \log q)$ complexity of the outer low-density parity-check decoder.

A more important advantage of increasing q and n comes from the fact that the uncertainty in the bits of $\boldsymbol{s}$ is highly correlated over short distances. As an extreme example, take the received string "0101010101 ...," and assume that the decoder has determined that there is a 50% probability that the first symbol was a spurious insertion, but the following stream is reliable. The entropy of any single bit is then 1 (i.e., it might be in either of two equally likely states), but the joint entropy of groups of n following bits is also only 1, for all n. In this artificial example, there are only two possible values each q-ary symbol can take, regardless of the size of q. If P_{i} and P_{d} are small, then a region of uncertainty of a characteristic width surrounds the position of insertion/deletion events. Outside these regions symbols are more strongly determined.

If q is fixed, then increasing n reduces the density of the sparse vector $\boldsymbol{s}$: asymptotically the density of s is $(q-1)/nq$. For moderate values of n this reduction improves decoding, but in the limit of large n the increased number of synchronization errors present in regions of length n outweighs the benefit of sparser $\boldsymbol{s}$. Eventually, the entropy increases with increasing n rendering the estimation of the q-ary symbols more difficult. This effect is shown most clearly in Fig. 13 for GF (2) and GF (4).

The conditional entropy $H(\boldsymbol{d}|\boldsymbol{r})$ (to be precise, the entropy of the normalized symbol-by-symbol likelihood functions $P(\boldsymbol{r}|d_i)$) can be used to infer a lower bound on the capacity of the channel using watermark codes. This entropy depends on the choice of n and q, and the channel parameters. Fig. 14 shows the average conditional entropy $H(\boldsymbol{d}|\boldsymbol{r})$ as a function of the channel insertion and deletion parameters.

The entropy surface identifies upper bounds on the rate of the low-density parity-check code for reliable communication, for a given channel and watermark code. Given the success of low-density parity-check codes for Gaussian channels [16], [27], we conjecture that this upper bound is closely approachable in the limit of long block lengths and well designed low-density parity-check codes. If this conjecture holds then by subtracting the conditional entropy in Fig. 14 from 1 we obtain lower bounds on the capacity, C_{eff}, of the *effective channel* seen by the outer code after watermark decoding.

A lower bound on the capacity of the insertion/deletion channel proper may be obtained by multiplying C_{eff} by the rate of the watermark code. Fig. 15 compares this empirical lower bound on capacity with the bounds calculated by Ullman and Zigangirov [12], [13]. The empirical lower bounds improve on Zigangirov's lower bound at noise levels above $P_{\mathrm{d}} = P_{\mathrm{i}} = 0.1$. Furthermore, the empirical lower bounds suggest that watermark codes enable practical communication at rates considerably greater than the computational cutoff rate.

While these empirical capacity bounds are speculative, we believe that, in the absence of a known capacity result, they give a useful indication of the limits of watermark codes.

ACKNOWLEDGMENT

The authors would like to thank V. I. Levenshtein and D. Forney for useful discussions, and H. Ferreira and D. Neuhoff for suggesting relevant references.

REFERENCES

[1] F. H. C. Crick, J. S. Griffith, and L. E. Orgel, "Codes without commas," *Proc. Nat. Acad. Sci. USA*, vol. 43, pp. 416–421, 1957.

[2] S. W. Golomb, B. Gordon, and L. R. Welch, "Comma-free codes," *Can. J. Math.*, vol. 10, no. 2, pp. 202–209, 1958.
[3] T. R. Hatcher, "On a family of error-correcting and synchronizable codes," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 620–624, Sept. 1969.
[4] J. J. Stiffler, "Comma-free error-correcting codes," *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 107–111, Jan. 1965.
[5] S. E. Tavares and M. Fukada, "Matrix approach to synchronization recovery for binary cyclic codes," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 93–101, Jan. 1969.
[6] F. F. Sellers Jr, "Bit loss and gain correction code," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 35–38, Jan. 1962.
[7] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Sov. Phys.—Dokl.*, vol. 10, no. 8, pp. 707–710, Feb. 1966.
[8] L. Calabi and W. E. Hartnett, "A family of codes for the correction of substitution and synchronization errors," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 102–106, Jan. 1969.
[9] E. Tanaka and T. Kasai, "Synchronization and substitution error-correcting codes for the Levenshtein metric," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 156–162, Mar. 1976.
[10] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion," *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 766–769, Sept. 1984.
[11] A. S. J. Helberg, "Coding for the Correction of Synchronization Errors," Ph.D. dissertation, Fac. Eng. , Rand Afrikaans Univ., Nov. 1993.
[12] J. D. Ullman, "On the capabilities of codes to correct synchronization errors," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 95–105, Jan. 1967.
[13] K. Sh. Zigangirov, "Sequential decoding for a binary channel with drop-outs and insertions," *Probl. Pered. Inform.*, vol. 5, no. 2, pp. 23–30, 1969.
[14] J. M. Wozencraft and I. M. Jacobs, *Principles of communication engineering*. New York: Wiley, 1965.
[15] L. J. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," *IEEE Trans. Inform. Theory*, vol. 45, pp. 2552–2557, Nov. 1999.
[16] M. C. Davey and D. J. C. MacKay, "Low density parity check codes over GF (q)," *IEEE Commun. Lett.*, vol. 2, pp. 165–167, June 1998.
[17] D. J. C. MacKay and M. C. Davey, "Evaluation of Gallager codes for short block length and high rate applications," in *Codes, Systems and Graphical Models*, B. Marcus and J. Rosenthal, Eds. New York: Springer-Verlag, 2000, vol. 123, IMA Volumes in Mathematics and its Applications, pp. 113–130.
[18] M. C. Davey, "Error-correction using low-density parity-check codes," Ph.D. dissertation, Univ. Cambridge, Cambridge, U.K., Dec. 1999.
[19] N. Kashyap and D. L. Neuhoff, "Codes for data synchronization and timing," in *Proc. 1999 IEEE Information Theory and Communications Workshop*, 1999, IEEE Catalog Number 99EX253, pp. 63–65.
[20] D. Sankof and J. B. Kruskal, Eds., *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparisons*. Reading, MA: Addison-Wesley, 1983.
[21] K. Kukich, "Techniques for automatically correcting words in text," *Comput. Surv.*, vol. 24, pp. 377–439, 1992.
[22] P. Hall and G. Dowling, "Approximate string matching," *Comput. Surv.*, vol. 12, pp. 381–402, 1980.
[23] M. S. Waterman, *Introduction to Computational Biology*. London, U.K.: Chapman and Hall, 1995.
[24] R. G. Gallager, "Low density parity check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan 1962.
[25] D. J. C. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.
[26] F. Jelinek, *Statistical Methods for Speech Recognition*. Cambridge, MA: MIT Press, 1998.
[27] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Information Theory*, vol. 47, pp. 599–618, Feb. 2001.
[28] P. A. H. Bours, "Codes for correcting insertion and deletion errors," Ph.D. dissertation, Eindhoven Tech. Univ., Eindhoven, The Netherlands, June 1994.
[29] M. Morse, "Recurrent geodesics on a surface of negative curvature," *Trans. Amer. Math. Soc.*, vol. 22, pp. 84–110, 1921.
[30] M. A. Ribeiro, "Optimal bit-level synchronization strategy for digital magnetic recorders," in *Proc. 14th Symp. Information Theory in the Benelux*, 1993, pp. 222–227.
[31] E. A. Ratzer. (2000, May) Reliable communication over insertion– deletion channels. University of Cambridge. [Online]. Available: http://wol.ra.phy.cam.ac.uk/is/papers/ear-indel.ps.gz
[32] R. G. Gallager, "Sequential decoding for binary channels with noise and synchronization errors," unpublished, Lincoln Lab Rep. 25 G-2, 1961.