Joint source-channel decoding: a cross layer perspective

Pierre Duhamel and Michel Kieffer
Laboratoire des Signaux et Systèmes
CNRS – Supélec – Univ Paris-Sud

11 octobre 2007
Acknowledgements

This work has been partly supported by

- RNRT COSOCATI, VIP, DIVINE
- Alcatel, CNES
- ReX NEWCOM

Joint work with

- Hang Nguyen, Rachel Chiang, Chang-Ming Lee, Salma Ben-Jamaa, Cédric Marin, Khaled Bouchireb, Danish Naem, Galina Sabeva...
- Amine Bouabdallah, Jérôme Lacan
Organisation of these two talks

• Introduction, context
• Why joint source and channel decoding?
• Identifying residual redundancy
• Taking this redundancy into account
• Towards practical implementations
• Taking the network into account
• Open challenges
• Joint arithmetic decoding
1 Introduction

1.1 Context of the work

Robust transmission of multimedia contents over

- wireless channels
- mixed wired-wireless channels

Channel introduces

- data losses (wired part)
- transmission errors (wireless part)
1.2 Solutions currently used

Combination of

- packetization
  → limits effect of lost data

- acknowledgement mechanisms (ACK/NACK)
  → erroneous packets are reemitted (ARQ)

But

- ARQ not always possible
Example 1: visophony over IP (RNRT VIP)

Characteristics

- mixed wired-wireless channels
- strong delay constraints
- limited ARQ
Example 2: video broadcasting (CNES SDMB, Alcatel TVMSL)

Characteristics

- wireless channels
- limited delay constraints
- no ARQ
Example 3: Multimedia multicasting (RNRT DIVINE)

Characteristics

- mixed wired-wireless channels
- limited delay constraints
- limited ARQ
1.3 The source of interest

Two video coders considered: H.263 [ITU95] and H.264 [II03]
Basic processing unit: macroblock (16 × 16 pixels)

1. Motion compensation
   ← only for INTER macroblocks → texture macroblocks

2. Texture macroblocks divided in blocks (8 × 8 pixels)

3. For each texture block
   (a) DCT
   (b) quantization
   (c) zig-zag scan
   (d) variable-length encoding
For H.263

- (run, level, last) encoding of texture data after zig-zag scan
- Huffman-based variable-length codes (VLC)

For H.264

- size of the blocks may be adjusted
- motion compensation at a block level
- context-adaptive VLC

or

- context-adaptive binary arithmetic coding
1.4 Network adaptation - application layer point of view

H.263+ does not provide interface
→ ad hoc solutions

H.264 provides basic interfacing unit
→ Network abstraction layer (NAL) [II03]

Macroblocks grouped in slices

- each encoded independently
- one or several slices per network packet

Short header added to slice or group of slices
↓
Transmitted to lower protocol layer
2 Why source and channel joint decoding?

Source coding: reduction of uncontrolled redundancy present in source

Channel coding: introduction of structured redundancy

Shannon’s joint source-channel coding theorem [Sha48, CT91]:

Source and channel separately optimized

\[ \iff \]

Joint optimization

\[ \downarrow \]

Decades of separation
But

- channel codes efficient for very long block length
  → residual decoding errors in schemes with constrained complexity,
- source codes designed for error-free channels
  → prone to uncorrected channel errors.

Decoded H.263+ frame at output of AWGN channel with SNR = 10 dB
Solution suggested by Shannon [Sha48]:

... However, any redundancy in the source will usually help if it is utilized at the receiving point. In particular, if the source already has a certain redundancy and no attempt is made to eliminate it in matching to the channel, this redundancy will help combat noise. For example, in a noiseless telegraph channel one could save about 50% in time by proper encoding of the messages. This is not done and most of the redundancy of English remains in the channel symbols. This has the advantage, however, of allowing considerable noise in the channel. A sizable fraction of the letters can be received incorrectly and still reconstructed by the context. In fact this is probably not a bad approximation to the ideal in many cases...

Main idea of source and channel joint decoding
3 Identifying redundancy

3.1 Redundancy due to the spelling of codewords

Mainly for VLCs.

Consider the VLC

\[ C = \{ c_1, \ldots, c_M \} \]

when the Kraft inequality [CT91] is strict

\[ \sum_{i=1}^{M} 2^{-\ell(c_i)} < 1, \]

\( C \) is not complete.
Example: With $\mathcal{C} = \{0, 11, 101\}$, a sequence like

$$1110011$$

could not be generated.

Set of sequences of $N$ bits: $\mathbb{B}^N$

Set of sequences of $N$ bits consistent with the VLC spelling: $\mathbb{S}_N$

$$\mathbb{S}_N \subseteq \mathbb{B}^N.$$ 

Redundancy (per block) due to incomplete code

$$R_S = N - \log_2 |\mathbb{S}_N| \text{ bits}$$

VLC for H.263 is not complete.
3.2 Redundancy due to source–VLC probability mismatch

Consider

- a source $X$, with $p(x_i) = \Pr(X = x_i)$, $i = 1 \ldots M$
- a Shannon VLC $C = \{c_1, \ldots, c_M\}$ designed for $q(x_i)$, i.e.,

$$\ell(c_i) = \lceil -\log q(x_i) \rceil.$$  

Then

$$H(p) + D(p||q) \leq E_p(\ell(X)) \leq H(p) + D(p||q) + 1.$$  

Redundancy (per source symbol) due to mismatch [CT91]

$$R_M = D(p||q) = \sum_x p(x) \log_2 \frac{p(x)}{q(x)} \text{ bits}.$$  

Very difficult to take into account.
3.3 Redundancy for a Markov source

Markov source encoded with a VLC designed for a memoryless source

\[\downarrow\]

Redundancy left in the bitstream

Redundancy identified in [GSC00, TK03a].
3.4 Redundancy due to the semantic of the source coder

Encoded bitstream has to satisfy many rules defined by the standard

\[ \Downarrow \]

Standard specific redundancy.

**H.263 coding of a texture block**

DCT, quantization, zigzag scan \( \implies \) 64 coefficients

\[ (12, 0, 0, 0, 4, 0, 0, -6, 0, 0, 1, 0 \ldots 0) \]

encoded with triples \((Run, Level, Last)\)

- **run**: number of zeros before the next non-zero coefficient
- **level**: value of this coefficient
- **last**: flag indicating whether it is the last non-zero coefficient of the block.

\( (0, 12, 0), (3, 4, 0), (2, -6, 0), (2, 1, 1) \)
For each block (8 × 8 pixels)

- VLCs represent at most 64 transformed coefficients,
- last VLC has to be such that Last = 1.

Evaluation of the redundancy due to the syntax [ND03]
Some H.264 coding rules leading to residual redundancy

Examples [SBJKD06]:

Coder is more efficient $\implies$ less redundancy
Joint decoding tools have to be adapted
3.5 Redundancy due to packetization

Compressed data put into packets

- implicit in H.263,
- part of the standard in H.264 (NALU) ⇒ more efficient packetization

Redundancy present due to

- compressed bitstream, which cannot be cut at any position
  - end of packet = end of block/slice
- independance between packets to allow robustness to lost packets
- side information present in packet headers

⇒ many constraints on the bitstream
4 Evidencing the structure of the redundancy

4.1 Notations

Discrete random variable $X$ with

• alphabet $\mathcal{X} = \{0 \ldots M - 1\}$

• probability distribution function $p(x) = \Pr(X = x)$.

Variable-length code $\mathcal{C}$ such that $x \rightarrow c(x)$

• $\ell_{\min} = \min_{c \in \mathcal{C}} \ell(c)$ and $\ell_{\max} = \max_{c \in \mathcal{C}} \ell(c)$
Packet of $K$ source symbols $x = (x_1 \ldots x_K)$

⇓

Packet of $K$ VLC codewords $c = (c_1 \ldots c_K)$

$\iff$

Packet of $N$ bits $b = (b_1 \ldots b_N)$

⇓

Packet of $N$ channel outcomes $y = (y_1 \ldots y_N)$
4.2 Estimators

Sequence MAP decoding

\[ \hat{x}_{1:K} = \arg \max_{x_1 \ldots x_K} \Pr(x_1 \ldots x_K | y_1 \ldots y_N). \quad (1) \]

Symbol-by-symbol MAP decoding

\[ \hat{x}_k = \arg \max_{x_k} \Pr(X_k = x_k | Y_{1:N} = y_{1:N}). \quad (2) \]

Computation of (1) or (2)

- optimal on trellises,
- sub-optimal using sequential algorithms.
Trellis represents all successions of VLC codewords satisfying some constraints.

More constraints $\implies$ higher complexity

Tradeoff between efficiency and complexity
4.3 Bit-clock trellis

Consider \( C = \{00, 11, 101, 010, 0110\} \)

Tree representation of the VLC codeword
Bit-clock trellis of the VLC codeword [BH98]

Structures redundancy due to spelling of VLC

One-dimensionnal state $S_n = m$, with $m \in S = \{R, I1, I2, I3, I4, I5\}$
Sequence 00, 11, 00
Sequence 101, 010
Assume that $N = 6$ is known

Bit-clock trellis with length constraint $N = 6$
Bit-clock trellis suited for evaluating bit-level APPs

\[ \Pr (B_n = b | Y_{1:N} = y_{1:N}) , \]

e.g., with adaptation of BCJR algorithm [BCJR74] by [BH98].

Notations and hypotheses:

- \( p_n (m|m') = \Pr (S_n = m | S_{n-1} = m') \) transition probability on VLC tree
- \( q_n (b|m', m) = \Pr (B_n = b | S_{n-1} = m', S_n = m) \) probability of the output bits
- channel assumed memoryless and stationary

\[ \Pr (Y_{1:n} | B_{1:n}) = \prod_{j=1}^{n} R (Y_j | B_j) \]
To evaluate $\Pr (B_n = b | Y_{1:N} = y_{1:N})$, one evaluates first
$\Pr (B_n = b, Y_{1:N} = y_{1:N})$.

One may write

$$\Pr (B_n = b, Y_{1:N} = y_{1:N}) = \sum_{m',m} \Pr (B_n = b | S_{n-1} = m', S_n = m, Y_{1:N} = y_{1:N}) \cdot \Pr (S_{n-1} = m', S_n = m, Y_{1:N} = y_{1:N})$$

Output bit $b$ determined by transition between states

$$\Pr (B_n = b, Y_{1:N} = y_{1:N}) = \sum_{m',m} q_n (b|m',m) \Pr (S_{n-1} = m', S_n = m, Y_{1:N} = y_{1:N}).$$

One has thus to evaluate

$$\sigma_n (m',m) = \Pr (S_{n-1} = m', S_n = m, Y_{1:N} = y_{1:N}).$$
One may write $\sigma_n (m', m)$ as

$$\sigma_n (m', m) = \alpha_{n-1} (m') \gamma_n (m', m) \beta_n (m),$$

with **forward** recursion for $\alpha$

$$\alpha_n (m) = \sum_{m'} \Pr (S_{n-1} = m', S_n = m, Y_{1:n}) = \sum_{m'} \alpha_{n-1} (m') \gamma_n (m', m),$$

and **backward** recursion for $\beta$

$$\beta_n (m) = \sum_{m'} \Pr (S_{n+1} = m', Y_{n+1:N} | S_n = m) = \sum_{m'} \beta_{n+1} (m') \gamma_{n+1} (m, m'),$$

with

$$\gamma_n (m', m) = \sum_b p_n (m', m) q_n (b|m', m) R (Y_n = y_n | B_n = b).$$
Initial conditions

\[ \alpha_0 (R) = 1, \alpha_0 (m) = 0, m \neq R, \]
\[ \beta_N (R) = 1, \beta_N (m) = 0, m \neq R. \]
Assume now that \( N \) and \( K \) are known → previous trellis has to be adapted to take this additional constraint into account.

Two-dimensional state \( S_n = (m, k) \), with \( m \in S = \{R, I1, I2, I3, I4, I5\} \) and \( k = 0 \ldots K \)

3D trellis representation more compact ↓ Recursions of the BCJR algorithm still possible
Example:

\[ N = 6, \ K = 2 \]

2D trellis representation

3D trellis representation

Each dimension takes a constraint into account
4.4 Symbol-clock trellis

Consider $\mathcal{C} = \{1, 01, 00\}$

Symbol-clock trellis representation, $K = 4$, $N = 6$ [BH00a, KB00]

One-dimensionnal state $S_k = n$, with $n = 0 \ldots N$

Parallel transitions, bit transitions on the VLC tree are lost
Sequence 1, 00, 01, 1
Symbol-clock trellis suited for evaluating symbol-level APPs

\[ \Pr (X_k = i | Y_{1:N} = y_{1:N}) , \]

e.g., with adaptations of BCJR algorithm [BCJR74] by [BH00a].

Notations and hypotheses:

- \( p_k (n | n') = \Pr (S_k = n | S_{k-1} = n') \) transition probability on VLC tree
- \( q_k (x | n', n) = \Pr (X_k = x | S_{k-1} = n', S_k = n) \) probability of the input symbol
- channel assumed memoryless and stationary

\[ \Pr (Y_{1:n} | B_{1:n}) = \prod_{j=1}^{n} R (Y_j | B_j) \]
Evaluation of $\Pr(X_k = i | Y_1^N = y_1^N)$ may be done as

$$
\Pr(X_k = i | Y_1^N = y_1^N) = \frac{1}{\Pr(Y_1^N = y_1^N)} \sum_{n \in \mathcal{N}_k} \sum_{n' \in \mathcal{N}_{k-1}} \alpha_{k-1}(n') \gamma_i(y_{n' + 1:n}, n', n) \beta_k(n)
$$
Requires again

- **forward** recursion for $\alpha_k$

\[
\alpha_k (n) = \sum_{n' \in \mathcal{N}_{k-1}} \sum_{i=0}^{M-1} \alpha_{k-1} (n') \gamma_i (y_{n'+1:n}, n', n),
\]

- **backward** recursion for $\beta_k$

\[
\beta_k (n) = \sum_{n' \in \mathcal{N}_{k+1}} \sum_{i=0}^{M-1} \beta_{k+1} (n') \gamma_i (y_{n'+1:n}, n, n'),
\]

- evaluation of

\[
\gamma_i (y_{n'+1:n}, n', n) = q_k (x|n', n) \cdot \Pr (Y_{n'+1:n} | X_k = x) \cdot p_k (n|n'),
\]

with

\[
\Pr (Y_{n'+1:n} | X_k = i) = \prod_{j=1}^{\ell (c_i)} p (y_{n'+j} | c_{i,j}).
\]
4.5 Taking more constraints into account

Previous trellises take into account

- $N$, number of bits of the packet,
- $K$, number of VLC codewords in the packet.

More sophisticated trellises have been proposed

- 3D trellis for representing sources with memory [TK03b, JCS05],
- 4D trellis for representing source semantics and packetisation [LKD05]

Each constraint adds a new dimension.
Adding constraints imposed by a channel code

- **Optimal solution:**
  decode on trellis combining code constraints and VLC constraints

- **Suboptimal solution:**
  perform turbo-like decoding [BH00b, TK03a, JCS05]
Note that

- channel code use bit-clock trellis
  \( \rightarrow \) iterations at bit-level easier

- convergence analysis by EXIT charts [TB01, JV05]
4.6 Fighting against complexity

Simplification of trellises

- grouping parallel branches [MKKD05],

- splitting the constraints into two groups and performing turbo-like decoding [LKD05],

- using state aggregation and modulo trellises [JMG05]

Constraint on $K \implies$ constraint on $K \mod K_0$
4.7 Fighting against complexity: sequential decoding

To reduce complexity, all paths on the trellis are no more explored

- breadth-first approach: $M$-algorithm
- depth-first approach: stack algorithm

For more details, see, e.g., [AM91].

Assume that a video coder generates a packet of $N = 5$ bits

$$\mathbf{u}_{1:N} = [1, 1, 0, 0, 0]$$

which is BPSK modulated $\mathbf{x} = f(\mathbf{u})$ and transmitted over an AWGN channel. Channel output

$$\mathbf{y}_{1:N} = [1.53, 1.22, -1.92, -3.17, -1.06].$$
MAP estimate of $\mathbf{u}_{1:N}$ using $\mathbf{y}_{1:N}$

$$
\hat{\mathbf{u}}_{1:N} = \arg \max_{\mathbf{v}_{1:N} \in \mathbb{S}} p(\mathbf{v}_{1:N}|\mathbf{y}_{1:N})
$$

$$
= \arg \max_{\mathbf{v}_{1:N} \in \mathbb{S}} \frac{p(\mathbf{y}_{1:N}|\mathbf{v}_{1:N}) p(\mathbf{v}_{1:N})}{p(\mathbf{y}_{1:N})}.
$$

(3)

where

$$
\mathbb{S} \subset \{0, 1\}^N
$$

is the set of all $N$ bits packets compliant with syntax, semantic... of the source coder.

For an AWGN channel,

$$
\hat{\mathbf{u}}_{1:N} = \arg \max_{\mathbf{v}_{1:N} \in \mathbb{S}} \exp \left( -\frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - f(v_i))^2 \right) \frac{p(\mathbf{v}_{1:N})}{p(\mathbf{y}_{1:N})}
$$

(4)

Obtaining $\hat{\mathbf{u}}_{1:N}$ requires at most $|\mathbb{S}|$ metric evalutations.
Sequential decoders perform bit-by-bit estimation of $\mathbf{u}_{1:N}$ by maximizing a metric derived from

$$\hat{\mathbf{u}}_{1:N} = \arg \max_{\mathbf{v}_{1:N} \in S} \exp \left( -\frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - f(v_i))^2 \right) \frac{p(\mathbf{v}_{1:N})}{p(\mathbf{y}_{1:N})}$$

(5)

by taking the $-\log(\cdot)$

$$\mathcal{M}(\mathbf{y}_{1:n}, \mathbf{v}_{1:n}) = \sum_{i=1}^{n} (y_i - f(v_i))^2 - 2\sigma^2 \log p(\mathbf{v}_{1:n}) + 2\sigma^2 \log p(\mathbf{y}_{1:n}).$$

(6)

This metric has to be maximized.
4.7.1 $M$–algorithm

Only sequences of the same length are compared

⇒

New metric to be minimized

$$
M_M(y_{1:n}, v_{1:n}) = \sum_{i=1}^{n}(y_i - f(v_i))^2 - 2\sigma^2 \log p(v_{1:n}).
$$

- Algorithm initialized with 2 sequences of 1 bit (0 and 1), which metric is evaluated

- At iteration $n$,
  - expansion of all sequences with 1 bit (0 and 1)
  - evaluation of the metrics
  - sequences are sorted
  - only the $M$ best sequences (with lowest metric) are kept.

$M$, tunes compromise between efficiency and complexity
For

\[ y_{1:N} = [1.53, 1.22, -1.92, -3.17, -1.06], \]

with \( M = 4 \), and all input sequences equally likely (second term in \( M_M \) not considered), one gets

\[
\begin{array}{c|c|c|c}
6.39 & 0 & 0.28 & 1 \\
0.28 & 1 & 6.39 & 0 \\
\end{array}
\]

Iteration \( n = 2 \)

\[
\begin{array}{c|c|c|c|c|c}
5.20 & 1 & 0 & 0.33 & 1 & 1 \\
11.32 & 0 & 0 & 5.20 & 1 & 0 \\
0.33 & 1 & 1 & 6.44 & 0 & 1 \\
6.44 & 0 & 1 & 11.32 & 0 & 0 \\
\end{array}
\]
Iteration $n = 3$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.18</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6.05</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7.29</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8.87</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12.17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>13.74</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>14.98</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19.86</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

$\implies$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.18</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6.05</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7.29</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8.87</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11.18</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5.05</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7.29</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8.87</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13.74</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>14.98</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19.86</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Iteration $n = 4$

<table>
<thead>
<tr>
<th></th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
<th>Column 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.89</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.77</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.58</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.57</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23.45</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24.69</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26.26</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\Rightarrow$

<table>
<thead>
<tr>
<th></th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
<th>Column 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.89</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.77</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.58</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23.45</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24.69</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26.26</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Iteration $n = 5$

<table>
<thead>
<tr>
<th></th>
<th>1 1 0 0 0 0</th>
<th>1 1 0 0 0 0</th>
<th>1 1 0 0 0 0</th>
<th>1 1 0 0 0 0</th>
<th>1 1 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.89</td>
<td>1 1 0 0 0 0</td>
<td>1 1 0 0 0 0</td>
<td>1 1 0 0 0 0</td>
<td>1 1 0 0 0 0</td>
<td>1 1 0 0 0 0</td>
</tr>
<tr>
<td>10.77</td>
<td>1 0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td>12.01</td>
<td>0 1 0 0 0 0</td>
<td>0 1 0 0 0 0</td>
<td>0 1 0 0 0 0</td>
<td>0 1 0 0 0 0</td>
<td>0 1 0 0 0 0</td>
</tr>
<tr>
<td>13.58</td>
<td>1 1 1 0 0 0</td>
<td>1 1 1 0 0 0</td>
<td>1 1 1 0 0 0</td>
<td>1 1 1 0 0 0</td>
<td>1 1 1 0 0 0</td>
</tr>
<tr>
<td>10.13</td>
<td>1 1 0 0 0 1</td>
<td>1 1 0 0 0 1</td>
<td>1 1 0 0 0 1</td>
<td>1 1 0 0 0 1</td>
<td>1 1 0 0 0 1</td>
</tr>
<tr>
<td>15.01</td>
<td>1 0 0 0 0 1</td>
<td>1 0 0 0 0 1</td>
<td>1 0 0 0 0 1</td>
<td>1 0 0 0 0 1</td>
<td>1 0 0 0 0 1</td>
</tr>
<tr>
<td>16.24</td>
<td>0 1 0 0 0 1</td>
<td>0 1 0 0 0 1</td>
<td>0 1 0 0 0 1</td>
<td>0 1 0 0 0 1</td>
<td>0 1 0 0 0 1</td>
</tr>
<tr>
<td>17.82</td>
<td>1 1 1 0 0 1</td>
<td>1 1 1 0 0 1</td>
<td>1 1 1 0 0 1</td>
<td>1 1 1 0 0 1</td>
<td>1 1 1 0 0 1</td>
</tr>
</tbody>
</table>

Estimate $\hat{u}_{1:N}$ is then

$$\hat{u}_{1:N} = [1, 1, 0, 0, 0].$$
4.7.2 Stack algorithm

Breadth-first approach, expand the best sequence first [Zig66, Jel69].

Metric to be minimized

\[ M(y_{1:n}, v_{1:n}) = \sum_{i=1}^{n} (y_i - f(v_i))^2 - 2\sigma^2 \log p(v_{1:n}) + 2\sigma^2 \log p(y_{1:n}) \]

complex to evaluate. Use approximation proposed by Fano [Fan63]

\[ p(y_{1:n}) = 2^{-n}. \]
New metric to be minimized

$$\mathcal{M}_S (y_{1:n}, v_{1:n}) = \sum_{i=1}^{n} (y_i - f(v_i))^2 - 2\sigma^2 \log p(v_{1:n}) - 2\sigma^2 n \log 2$$

- Algorithm initialized with 2 sequences of 1 bit (0 and 1), which metric is evaluated

- At iteration $n$,
  - expansion of the best sequence with 1 bit (0 and 1)
  - evaluation of the metrics
  - sequences are sorted.

Number of sequences kept may be limited to bound computing time.
For

\[ y_{1:N} = [1.53, 1.22, -1.92, -3.17, -1.06], \]

with all input sequences equally likely \((p(v_{1:n}) = 2^{-n})\), one gets

\[
\mathcal{M}_S(y_{1:n}, v_{1:n}) = \sum_{i=1}^{n} (y_i - f(v_i))^2
\]

and

\[
\begin{array}{cc|ccc}
6.39 & 0 & 0.28 & 1 \\
0.28 & 1 & 6.39 & 0 \\
\end{array} \implies \\
\begin{array}{cc|ccc}
5.20 & 1 & 0 & 0.33 & 1 & 1 \\
0.33 & 1 & 1 & 5.20 & 1 & 0 \\
6.39 & 0 & 6.39 & 0 \\
\end{array}
\]

Iteration \(n = 2\)
Iteration \( n = 3 \)

\[
\begin{array}{c|ccc}
& 1 & 1 & 0 \\
1.18 & 1 & 1 & 0 \\
8.87 & 1 & 1 & 1 \\
5.20 & 1 & 0 \\
6.39 & 0 \\
& 8.87 & 1 & 1 & 1 \\
\end{array}
\Rightarrow
\begin{array}{c|ccc}
& 1 & 1 & 0 \\
1.18 & 1 & 1 & 0 \\
5.20 & 1 & 0 \\
6.39 & 0 \\
& 8.87 & 1 & 1 & 1 \\
\end{array}
\]

Iteration \( n = 4 \)

\[
\begin{array}{c|ccc}
& 1 & 1 & 0 \\
5.89 & 1 & 1 & 0 \\
18.57 & 1 & 1 & 0 \\
5.20 & 1 & 0 \\
6.39 & 0 \\
& 8.87 & 1 & 1 & 1 \\
8.87 & 1 & 1 & 1 \\
\end{array}
\Rightarrow
\begin{array}{c|ccc}
& 1 & 1 & 0 \\
5.89 & 1 & 1 & 0 \\
5.20 & 1 & 0 \\
6.39 & 0 \\
& 8.87 & 1 & 1 & 1 \\
\end{array}
\]
**Iteration $n = 5$**

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6.05</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td>5.89</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13.74</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>6.05</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5.89</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6.39</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.39</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>8.87</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8.87</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>13.74</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>18.57</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>18.57</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Iteration $n = 6$

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.89</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.89</td>
</tr>
<tr>
<td>10.77</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6.05</td>
</tr>
<tr>
<td>6.05</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>6.39</td>
</tr>
<tr>
<td>6.39</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8.87</td>
</tr>
<tr>
<td>8.87</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>10.77</td>
</tr>
<tr>
<td>13.74</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>13.74</td>
</tr>
<tr>
<td>18.57</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>18.57</td>
</tr>
</tbody>
</table>

Estimate $\hat{u}_{1:N}$ is again

$$\hat{u}_{1:N} = [1, 1, 0, 0, 0].$$
4.7.3 Comparison

M-algorithm

- constant complexity
- may lead to decoding failure (when the correct solution is dropped)

Stack algorithm

- faster when the channel is good
- very slow when the channel is bad.
- may lead to decoding failure (when the number of kept lines is limited)
5 A first step for using JSCD in realistic situations

Methods described above usually require knowledge of

- number of bits in each macroblock
- number of symbols in each macroblock

Not available in practical situations.

Finding the structure of redundancy due to coder semantics much more complex to determine
5.1 When the structure of the redundancy may be evidenced

Illustration with H.263+ [LKD05], extension to H.264 with CAVLC by [BLB04, WKNAM04].

Summary of the properties that can be used:

At a block level:

1. after VLC encoding, number of coded coefficients has to be lower than $N_C = 64$
2. last VLC of a block has to be such that $\text{Last} = 1$

At a texture packet level (Number of blocks $N_B$ and of bits $N_b$ known):

1. total length of VLCs in packet: $N_b$ bits
2. last VLC of a packet such that $\text{Last} = 1$
3. Sum of VLCs representing Last = 1 has to be $N_B$
Structuring these constraints

At channel input

Packet of $N_b$ bits containing $N_B$ texture blocks

$$b = (b_1, \ldots, b_{N_b})^T$$

At channel output

$$y = (y_1, \ldots, y_{N_b})^T$$

ML or MAP estimation

$$\Downarrow$$

Maximization over the set of sequence of VLC satisfying 5 constraints
Packet of $N_b$ bits containing $N_B$ texture blocks, set of sequence of VLC satisfying 5 constraints

\[\downarrow\]

$4D$ bit-synchronised trellis – State $S_n = (k, r, \varepsilon)$

$n$ : current bit index

$k$ : current VLC index

$r$ : number of texture coefficients in current block

$\varepsilon$ : number of blocks already decoded

Prohibitive complexity
Suboptimal decoder

Decoding problem performed in two steps [LKD05]

1. Localization of texture block limits
   ← only constraints at a packet level are used
   ← 3D bit-time treillis – state $S_n = (k, \varepsilon)$

2. Decoding of each texture block
   ← constraints at a block level are used
   ← 3D bit-level treillis – state $S_n = (k, r)$

After step 1, blocks may contain more than 64 coefs

Sub-optimality
Applications

Sequence: foreman.qcif at 25 fps

Bitrate: about 150 kbps

Size of packets: about 1000 bits

Headers and motion vectors strongly channel coded

Considered decoders:

- \textit{HD}: standard hard-input decoder
- \textit{const-HD}: hard-input decoder satisfying constraints
- \textit{ML}: soft-inputs, constraints satisfying ML estimator
- \textit{MAP}: soft-inputs, constraints satisfying MAP estimator
- \textit{Known loc. MAP}: soft-inputs, constraints satisfying MAP estimator, limits of blocks known in each packet
For INTRA encoded pictures

Average packet data:

- $N_b = 758$ bits
- $N_B = 16$ blocks
For INTER encoded pictures

Average packet data:

- $N_b = 602$ bits
- $N_B = 29$ blocks
5.2 When the structure of the redundancy cannot be evidenced

Illustration with H.264 with CABAC.

- NALUs are independently decodable
- Decoding process assumes NALUs in decoding order
Transmission scheme

Consider $i$-th NALU $b_{1:N_i}^i$ (consists of $N_i$ bits)

Real-valued channel outcomes

$$y_{1:N_i}^i = \{y_1^i, \ldots, y_{N_i}^i\}$$
Residual redundancy in the H.264 NALUs

H.264 standard specifies the syntax of NALUs.

Set of sequences for the $i$-th NALU compliant with this syntax:

$$B_{N_i}^i \subset \{0, 1\}^{N_i}$$

Not all successions of bits are allowed in a NALU

$$\Downarrow$$

Redundancy left in the bitstream
Examples

- `forbidden_zero_bit` should be equal to 0
- `nal_unit_type` should range from 0 to 13
- etc.
Maximum-likelihood estimator

Vector of real-valued channel outcomes: \( y_{1:N_i}^i \)

\[ + \]

H.264 syntax constraints: \( \mathcal{B}_{N_i}^i \)

\[ \Downarrow \]

Maximum-likelihood estimate for \( b_{1:N_i}^i \)

\[ \hat{b} = \arg \max_{b \in \mathcal{B}_{N_i}^i} p(y_{1:N_i}^i | b) \]

But \( \mathcal{B}_{N_i}^i \) not well-structured

\[ \Downarrow \]

Suboptimal solution necessary (\textit{M}-algorithm, stack algorithm)
Further simplification

Syntax check time consuming (based on H.264 decoder)

\[ \Downarrow \]

Perform syntax check every $K$ iterations.

When no sequence satisfies syntax $\implies$ concealment.
Concealment

Replace lost data, see [WZ98, HSKK99] for more details.

Main idea: use the fact that

- smooth changes between pictures
- smooth changes within picture
• When whole picture is lost
  ← replace with previous one
  ← perform temporal interpolation using motion vectors

• When parts of a picture are lost [SK95, WKNAM04]
  ← perform spatial interpolation
  ← perform spatial and temporal interpolation

• When motion vectors are lost [CCW97]
  ← perform spatial interpolation
  ← perform temporal interpolation
Example: concealment using spatial interpolation

Without and with concealment of lost NALUs
Experimental results

- Sequence foreman.cif
- IBPBPB..., one IDR frame every 16 frames
- NALU size limited to 250 bytes
  - facilitates decoding and concealment
  - encoder efficiency reduced.
# Evaluation of introduced redundancy

<table>
<thead>
<tr>
<th>NALU size</th>
<th>SNR-Y(dB)</th>
<th>Bitstream size</th>
<th>Extra kbits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500 bytes</td>
<td>34.3</td>
<td>2988 kbits</td>
<td></td>
</tr>
<tr>
<td>250 bytes</td>
<td>34.3</td>
<td>3328 kbits</td>
<td>340 (11%)</td>
</tr>
<tr>
<td>180 bytes</td>
<td>34.3</td>
<td>3513 bits</td>
<td>525 (17%)</td>
</tr>
<tr>
<td>120 bytes</td>
<td>34.3</td>
<td>3699 bits</td>
<td>711 (24%)</td>
</tr>
</tbody>
</table>

Bitstream size for the 299 encoded frames of Foreman.cif
SNR of the luminance, for a Gaussian channel of SNR of 11 dB
Hard
PSNR-Y = 24.6 dB

Soft, 1 test/NALU
PSNR-Y = 33.7 dB

Soft, 2 tests/NALU
PSNR-Y = 37.1 dB

Soft, 4 tests/NALU
PSNR-Y = 37.1 dB

Picture 96 of foreman.cif sent over an AWGN channel with SNR = 11 dB
6 The global picture

This was only a first step: recall the global OSI description of the communication layers.

6.1 General OSI layers (in short)

1. Application layer. the interface of the user with the network. In the case of video:
   - the video is packetized in NALU’s (Network Abstraction Layer Unit)
   - each NALU is packetized in an RTP packet,
   - and a RTP header is added

2. Network layer.
   (a) UDP. Controls the reliability of a link through flow control, segmentation/desegmentation, etc...
   - incoming RTP packets are encapsulated in UDP datagrams
• UDP header is added
• 16 bits checksum is added to the datagram (covers header and data. Allows integrity check)

(b) IP. Performs routing, might perform segmentation/desegmentation, reports errors.
• IP header added to the UDP datagram
• 16 bits checksum of the header added

3. MAC layer (access to medium). May be quite different in various systems (because the medium constraints can be very different)
6.2 Mac layer in WIFI

Known as Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA)

In CSMA/CA a Wireless node that wants to transmit performs the following sequence

1. Listen on the desired channel

2. If channel is idle (no active transmitters) it sends a packet

3. If channel is busy (an active transmitter), waits until transmission stops a (random) contention period. then go to 2

To improve efficiency additional features are employed

1. Positive Acknowledgement (ACK) -contention period starts after ACK should be received-

2. MAC level retransmission if no ACK received

3. Fragmentation: when BER too bad, large packets can be fragmented by the transmitter and reassembled by the receiver.
6.3 Mac layer in DVB-H

- data bursts episodically transmitted during time-slices (to save battery)
- Multi Protocol Encapsulation (MPE) to insert the IP datagrams into the MPEG Transport Stream (TS, common to all DVB services)
- technique of protection (i.e., Forward Error Correction applied to the MPE frames) to reinforce the transmitted service (described later)

Consequences:

Due to the very bursty nature of the transmission, deep fadings will be a nightmare. Various interleavers have to be used, at the cost of a large delay (which should not be too large anyway, due to zapping constraints). The corresponding problems are not considered here.
7 Which opens new possibilities

In the first part, we were searching for redundancy

as seen previously, there is a lot in the various layers... but may be it is not available where we would need it...

Important remarks :

• when doing JSCD, one assumes the measured data (PHY layer) to be available at the source decoder (Application Layer)

• this is supposed to be forbidden : layers should be independent (as much as possible)

• but it does not change anything in the standard: so, why not ?

• however, there is a real drawback if we imagine some methods which would require the transmission (over the channel, or the network) of quantities that are not currently transmitted.
7.1 CRC at MAC layer used to produce an erasure channel

In some standards, the CRC (Mac layer) is here, but is not used to produce an ACK/NACK decision. It is used for checking the validity of the decoding.

=> the channel now turns to an erasure channel (blockwise): there are three possible states: "0", "1", and "not received"

=> one can use codes designed for such channels: any code which has the MDS (Maximum Distance Separable) property is good.

=> one such code: RS. This has been implemented in DVB-H as Multi-protocol Encapsulation (MPE)

described on the next slide
7.1.1 MPE-FEC of the Mac layer in DVB-H
7.1.2 How is it used?

- IP packets are stored columnwise
- Parity check bits computed on each row are added as additional rows
- the resulting data are sent out column by column, sliced in MPEG2 transport streams which constitute the actual packets sent on the channel.
- if some packets are not ACKed, they can be recovered using the parity bits, as long as their number does not exceed the correction capacity.
- note that all headers and CRC’s of higher levels are also sent on the channel.
7.1.3 How is it tuned?

Obviously, some tuning is required:

- H264 generates NALU’s of varying size
- only the limit size of RTP packets may be controlled
- these RTP packets are sliced in MPEG2 transport streams, and the loss of one MPEG2-TS may result in the loss of one or several NALU’s
- the size of a NALU is thus an important parameter

Influence of the NALU size

- small NALU: poor compression (the probability estimates are re-initialized) and loss of efficiency (ratio headers vs useful data)
- large NALU: a loss of a single MPEG2-TS results in a loss of large amount of data.

Obviously: tradeoff, depending on the channel probability of error.
Proportion of information bits transmitted for various limit sizes of the NALUs
7.2 CRC used as additional data on the source frame

Another possibility: the packet actually sent on the channel contains a CRC. This packet usually carries part of a NALU as shown below.
Assumptions:

- all these data are available at the source decoder.
- AWGN channel.

Some MAC packets will have a correct CRC through hard decoding. They are not further processed. CRC data can be discarded.

Other ones processed by soft decoding of the data, given measurements of whole block (i.e. payload + CRC). (easy to say...)

- All algorithms for decoding block codes can be used
- Problem: we would like to be able to still use the source redundancy.
- Hence: sequential decoding algo.
- ”only” the computation of the metric remains.
• Problem: a naive algorithm would require a complexity exponential with the length of the payload.

• we could derive an algorithm with a complexity exponential with the length of the CRC (similar to Wolf trellis?) still too complex.

• followed by the derivation a less complex, suboptimal version (Cedric Marin, to be submitted).

Simulation context:

• H264

• CAVLC

• intra and inter coded packets are transmitted over an AWGN channel

• at MAC layer, packets are segmented in frames of 48 bytes and a CRC of 4 bytes is added (802.11 like)

• data are BPSK modulated.
Comments:

- the suboptimal algorithm is not too bad...
- the use of CRC can be combined with the source robustness
- they are useful in different Eb/No
• as illustrated below

• More important, the use of CRC ONLY as a source of robustness may be a very interesting option:
  – no need to feed the data to the source decoder (remains between close layers
  – the redundancy is better structured
  – should be checked in an iterative process with the channel decoder.
8 .... and new problems

Most problems come when you try to use several ideas at the same time: in particular, JSCD does not seem at first glance to be compatible with

- the ACK/NACK
- nor with the "erasure channel" use of the CRC
- especially the iterative processes (e.g. source decoder / channel decoder) which require going back and forth between layers.
8.1 ACK/NACK (CRC, in conjunction with robust decoding)

As described above, the CRC at MAC layer is used to check the correct reception of the packet.

However, JSCD may well be able to cope with this packet (due to the use of the other types of redundancies) and may result in good quality video.

\[=\] the CRC computation should be replaced by something else....

we are working at it....

the solution may well require involving higher layers...
8.2 RS, or additional redundancy in the bit stream? (MDP and robust decoding)

The use of CRC done in DVB-H (for introducing an erasure channel) seems to prevent the use of the CRC at source level.

Only a real soft decoding of the RS code would be compatible,

... or may be it can be considered as an additional source of redundancy (seems intricate...)

this remains a topic for further consideration
8.3 What is the best use of redundancy?

Back to the Shannon citation:

Question: is it always useful to remove redundancy in the bit stream (while trying to use it at the channel level?)

may be not in the case of video: as evidenced at several places previously, robustness due to source properties can be useful in regimes where the classical channel coders may even be harmful... the answer depends on the allowed BER in the bitstream (and error cancellation can do a lot of work...)

the question certainly will turn out to be the choice of the best repartition of redundancies at the various levels: source, CRC, channel coders.

This problem begins to be very large, hence the need for analytical tools.
One of the strongest difficulty is that we are dealing with very different types of structured and non structured redundancies, that are usually processed with very different means.

We describe below a study on the arithmetic coder, undertaken with tools that belong to the channel coding area.

This allowed to understand the efficiency of artificially introduced redundancy in the AC, and to optimize the precise way it is introduced.
9 Robust decoding of arithmetic codes

9.1 Introduction

Arithmetic coding [WNC87] replaces Huffman-based VLC

- JPEG 2000,
- H.263+, H.264

However, arithmetic coding (AC) is very sensitive to transmission errors

Development of Joint Source-Channel (JSC) techniques for AC-encoded data [BCI+97, Say99, CR00, PHS01, GG03, GCO05, DHS06].

Need for tools to compare analytically the efficiency of the proposed JSC techniques.
9.2 Arithmetic coding primer

Binary stationary source $X$ with

$$P(X = 0) = P_0 \text{ and } P(X = 1) = P_1 = 1 - P_0$$

generates $N$ outcomes

$$\mathbf{x} = (x_1, \ldots , x_N).$$

Arithmetic coding: associates a unique rational number $c(\mathbf{x})$ to $\mathbf{x}$ such that

- $c(\mathbf{x}) \in [0, 1[$,
- $N.H(X) \leq E[\ell(c(\mathbf{x}))] \leq N.H(X) + 2$
Infinite-precision arithmetic coder

\[ \text{low}_0 \]
\[ (0.00...)_2 \]

\[ 0.0 \]

\[ \text{high}_0 \]
\[ (0.11...)_2 \]

Arithmetic coding with \( P_0 = 1/5 \)
Infinite-precision arithmetic coder

\[
\begin{align*}
\text{low}_0 &= (0.00...)_2 \\
0.0 \\
0.2 \\
\text{high}_0 &= (0.11...)_2 \\
1.0
\end{align*}
\]

Arithmetic coding with \( P_0 = 1/5 \)
Infinite-precision arithmetic coder

Arithmetic coding with $P_0 = 1/5$
Infinite-precision arithmetic coder

Arithmetic coding with $P_0 = 1/5$
Infinite-precision arithmetic coder

Arithmetic coding with $P_0 = 1/5$
Infinite-precision arithmetic coder

\[
\begin{align*}
low_0 & = (0.00...)_2 \\
high_0 & = (0.11...)_2 \\
\text{low}_1 & = (0.00000...)_2 \\
\text{high}_1 & = (0.00100...)_2 \\
\text{low}_2 & = (0.00001...)_2 \\
\text{high}_2 & = (0.00110...)_2 \\
x_1 &= 0 \\
x_2 &= 1
\end{align*}
\]

Arithmetic coding with \( P_0 = 1/5 \)
Infinite-precision arithmetic coder

Arithmetic coding with $P_0 = 1/5$
Infinite-precision arithmetic coder

Arithmetic coding with $P_0 = 1/5$
Infinite-precision arithmetic coder

Arithmetic coding with $P_0 = 1/5$
Perform iteratively, starting from $[low_0, high_0] = [0, 1]$

1. division

$$[low_n, high_n]$$

$$\downarrow$$

$$[low_n, low_n + P_0 (high_n - low_n)], [low_n + P_0 (high_n - low_n), high_n]$$

2. selection

$$[low_{n+1}, high_{n+1}] = [low_n, low_n + P_0 (high_n - low_n)] \text{ if } x_{n+1} = 0,$$

$$[low_{n+1}, high_{n+1}] = [low_n + P_0 (high_n - low_n), high_n] \text{ if } x_{n+1} = 1.$$
After the $N$-th symbol has been encoded,

$$high_N - low_N = p(x).$$

Consider

$$\lambda(x) = \lceil - \log_2 p(x) \rceil + 1,$$

where $\lceil \cdot \rceil$ stand for rounding upwards.

Let

$$\mu(x) = \frac{high_N + low_N}{2}$$

and define

$$c(x) = \lfloor \mu(x) \rfloor_{\lambda(x)}$$

as $\mu(x)$ rounded downwards to the $\lambda(x)$-th decimal.
Then, one may prove that
\[ c(x) \in [\text{low}_N, \text{high}_N] \]
and
\[ E(\lambda(x)) = p(x) \left(\lceil -\log_2 p(x) \rceil + 1 \right) \]
satisfies
\[ E(\lambda(x)) \leq N.H(X) + 2. \]
Thus
\[ c(x) \text{ is a good representative for } x. \]
Other example

\[ \text{low}_0 \]
\[ (0.00...)_2 \]

\[ \text{high}_0 \]
\[ (0.11...)_2 \]

Arithmetic coding with \( P_0 = 1/5 \)
Other example

Arithmetic coding with $P_0 = 1/5$
Other example

Arithmetic coding with $P_0 = 1/5$
Other example

Arithmetic coding with $P_0 = 1/5$
Other example

Arithmetic coding with $P_0 = 1/5$
Other example

Arithmetic coding with $P_0 = 1/5$
Other example

Arithmetic coding with $P_0 = 1/5$
Other example

Arithmetic coding with $P_0 = 1/5$
Other example

Arithmetic coding with \( P_0 = 1/5 \)
Finite-precision arithmetic coder

Computations are performed on integers

\[ [0, 1] \rightarrow [0, 2^p], \]

where \( p \in \mathbb{N} \) is the finite precision of the AC.

To prevent intervals from getting too small

- when \([low_n, high_n] \subset [0, 2^{p-1}],\) 0 and \( \text{follow} \ 1 \) are emitted, size is doubled
- when \([low_n, high_n] \subset [2^{p-1}, 2^p],\) 1 and \( \text{follow} \ 0 \) are emitted, size is doubled
- when \([low_n, high_n] \subset [2^{p-2}, 3.2^{p-2}],\) \( \text{follow} \) is incremented, size is doubled.
Assuming that $\text{follow} = 2$

### Finite-precision arithmetic coding
9.3 Arithmetic codes as JSC codes

Robustness of AC against transmission errors achieved by introducing redundancy

- using a forbidden symbol \([\text{BCI}^+97]\), never used at the decoder
  - error detection capabilities
    - combination with ARQ \([\text{CR00}]\),
    - sequential decoding \([\text{PHS01}]\),
    - SISO MAP decoding \([\text{GCO05}]\).
- using gaps in the code interval \([\text{Say99}]\),
- quasi-arithmetic codes
  - leading to finite state machines \([\text{GG03}]\),
  - synchronisation markers \([\text{GG03}]\).

Which is the most efficient way to introduce redundancy?
Finite state machine $\Rightarrow$ trellis

Error-correcting performance characterized with distance properties on the trellis \cite{RC89}.

Distance properties evaluated for

- convolutional codes \cite{LC83},
- Variable Length Error Correcting (VLEC) codes \cite{BF95}.

Here

- evaluate distance properties of arithmetic codes
- optimize the redundancy in order to maximize distance.
9.4 Trellis-based arithmetic coding and decoding

From finite precision AC to trellis-based AC

- first idea by [HV92], AC by lookup tables,
- stochastic automaton by [GG03],
- 3D trellis taking into account the FS [DHS06].
AC interpreted as a state machine

Main idea:

\((low, high)\) takes a finite number of values.

If \(\text{follow}\) is bounded,

\((low, high, follow)\) takes a finite number of values.

State:

\((low, high, follow)\)

Transition:

One input symbol \(\longrightarrow\) zero to many output bits
Finite state machine obtained for $p = 4, P_0 = \frac{1}{5}, F_{max} = 1$

When forbidden symbol is present $\Rightarrow$ construction is similar
Distance properties more easily evaluated on output bit-synchronised trellis.

First step: eliminate mute transitions

Deleting mute transitions. (a) original state machine. (b) reduced state machine
Reduced state machine obtained for $p = 4, P_0 = \frac{1}{5}, F_{max} = 1.$
Bit-clock trellis for integer binary AC \((p = 4, P_0 = \frac{1}{5}, F_{max} = 1)\)

Encoding process: immediate.

Decoding process: requires Viterbi algorithm.
Second step: introduce intermediate states

$\leftarrow$ one output bit at each transition

Creating intermediate states
9.5 Performance analysis of trellis-based AC

Free distance

Free distance $d_{\text{free}}$: minimum Hamming distance between any two distinct pair of paths

- having the same length,
- stemming from a given node,
- converging to another node.

For linear codes,

$$d_{\text{free}} \text{ is minimum weight of paths diverging from and then remerging to the all zero path [LC83].}$$

For non-linear codes,

situation more complicated, all sequences have to be considered

$\implies$ combinatorial complexity.
Low-complexity evaluation of $d_{\text{free}}$

Iterative construction of three dimensional array $D_n$:

• for $y \neq z$, $D_n(x, y, z)$ is minimum distance between all pairs of paths
  – of $n$ bits,
  – starting at $x$,
  – ending respectively in $y$ and $z$,
  – never converging.

• for $y = z$, $D_n(x, y, y)$ is minimum distance between pairs of paths
  – of at most $n$ bits,
  – diverging from $x$
  – converging for the first time at $y$. 
More formally

- $C_n(x, y)$: set of all pairs of paths
  - of $n$ bits,
  - diverging from $x$,
  - converging for the first time at $y$.

- $D_n(x)$: set of all pairs of paths
  - of $n$ bits,
  - diverging from $x$,
  - never converging

One gets:

$$D_n(x, y, z) = \min_{(p^n_{xy}, q^n_{xz}) \in D_n(x)} d_H(p^n_{xy}, q^n_{xz}), \text{ if } y \neq z$$

$$D_n(x, y, y) = \min_{n' \leq n} \min_{(p^n_{xy}, q^n_{xy}) \in C_{n'}(x, y)} d_H(p^n_{xy}, q^n_{xy})$$

Then $d_{\text{free}}$ is

$$d_{\text{free}} = \min_{n, x, y} D_n(x, y, y).$$
Initializing $D_1$ for all transitions from $x \in S_{FSM}$ to $y \in S'_{FSM}$ one has

$$D_1(x, y, z) = \min_{t_{xy} \in T_{xy}, t_{xz} \in T_{xz}} d_H(t_{xy}, t_{xz}),$$

where $T_{xy}$ is the set of transitions from $x$ to $y$.

If $T_{xy}$ or $T_{xz}$ is empty, $d_H(T_{xy}, T_{xz}) = +\infty$. 
At each iteration, $D_n$ is deduced from $D_{n-1}$ as follows

$$D_n(x, y, z) = \min_{y', z' \in S_{FSM}'} \min_{t_{y'y}, t_{z'z} \in T_{y'y}, t_{z'z}} D_{n-1}(x, y', z') + d_H(t_{y'y}, t_{z'z}).$$

When $y = z$, $D_n(x, y, y)$ is updated

$$D_n(x, y, y) = \min\{D_{n-1}(x, y, y), D_n(x, y, y)\}.$$
Asymptotic error bounds

First coarse performance deduced from $d_{\text{free}}$

More precise one deduced from the distance spectrum.

Extension of results provided by [BF95] for VLEC.
Let $P_{er}^s$ be the error event probability at any source symbol position

$$P_{er}^s \leq \sum_{h=d_{free}}^{\infty} B_h P_h,$$

where

$$P_h = \frac{1}{2} \text{erfc} \frac{\sqrt{h \times E_b}}{\sigma \sqrt{2}} \text{ (AWGN)}$$

and

$$B_h = \sum_{n=1}^{\infty} \sum_{(p_n, q_n) \in G_n:} P(p_n) d_L (in(p_n), in(q_n)),$$

average Levenstein distance [Lev65] between input sequences of all converging pairs of paths whose output bistreams are at Hamming distance $h$.

$B_h$ still very complex to evaluate
9.6 Design of robust AC using distance properties

In [Say99, chap 1], current interval is partitioned at each iteration into three sets:

- two code intervals \( I_0 \) and \( I_1 \),
- a set \( I_{FS} \) allocated to the FS.

Generalization of the FS use Optimization parameters \( q_1 \) and \( q_2 \), as \( q_1 + q_2 + q_3 = 1 \).

Best pair of \( q_1 \) and \( q_2 \) defined as

\[
(q_1, q_2)_b = \arg \min \left\{ B_{d_{free}}(q_1, q_2) \mid d_{free}(q_1, q_2) = \max_{(q'_1, q'_2)} d_{free}(q'_1, q'_2) \right\}.
\]
9.7 Experimental results and discussion

Source with $P_0 = 1/8$.

Limitation of follow

<table>
<thead>
<tr>
<th>$F_{\text{max}}$</th>
<th>1</th>
<th>3</th>
<th>6</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional redundancy</td>
<td>5.19%</td>
<td>0.95%</td>
<td>0.60%</td>
<td>0.01%</td>
</tr>
<tr>
<td>$N_S$ in the exhaustive trellis</td>
<td>119</td>
<td>216</td>
<td>302</td>
<td>363</td>
</tr>
<tr>
<td>$N_S$ in the reduced trellis</td>
<td>26</td>
<td>37</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>$N_T$ in the reduced trellis</td>
<td>203</td>
<td>440</td>
<td>828</td>
<td>1404</td>
</tr>
</tbody>
</table>

Table 1: Effect of the choice of $F_{\text{max}}$ on the trellis features, $p = 6$, $P_\varepsilon = 0$
Effect of additional redundancy

FS between both subintervals.

Source sequences of 512 symbols, List Viterbi decoder.

BER and SER for different amounts of additional redundancy
## Optimization of the FS

<table>
<thead>
<tr>
<th>$P_\varepsilon$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>$w$</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>$q_1$</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>$q_2$</td>
<td>0.5</td>
<td>0.2</td>
<td>0.4</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>0.5</td>
</tr>
<tr>
<td>$d_{free}$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$A_{d_{free}}$</td>
<td>0.31</td>
<td>0.92</td>
<td>0.07</td>
<td>0.67</td>
<td>0.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.84</td>
<td>0.082</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.42</td>
<td>0.01</td>
</tr>
<tr>
<td>$B_{d_{free}}$</td>
<td>1.58</td>
<td>5.78</td>
<td>0.34</td>
<td>3.06</td>
<td>0.945</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.28</td>
<td>0.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.61</td>
<td>0.30</td>
</tr>
<tr>
<td>$N_S$</td>
<td>24</td>
<td>12</td>
<td>17</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$N_T$</td>
<td>122</td>
<td>77</td>
<td>81</td>
<td>47</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>35</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>29</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Effect of $P_\varepsilon$ on the trellis features, $p = 5, F_{max} = 1, P_0 = 0.2$. 
Figure 1: Comparison between two configurations of $I_{FS}$, for $P_\varepsilon = 0.2$ and $P_\varepsilon = 0.5$
10 Conclusion

This tutorial has many (too many?) targets

- Provide a review of the basic techniques that can be used for JSCD
- describe some of the difficulties that are encountered when trying to use JSCD in practical situations, with minimal context first
- provide information about precise, actual, contexts that would really require some cross layer tuning (at least), and would (hopefully) benefit from JSCD
- consider the use of JSCD in these contexts, showing many opportunities (as well as difficulties)
- provide some insight for further research
References


[GG03] T. Guionnet and C. Guillemot. Soft decoding and synchronization of arithmetic codes: Application to image


[RC89] M. Rouanne and D. J. Costello. An algorithm for computing the
distance spectrum of trellis codes. *IEEE Trans. on Information

Nr. 13099, EE Department, ETH Zurich, Switzerland, 1999.

[SBJKD06] G Sabeva, S. Ben-Jamaa, M. Kieffer, and P. Duhamel. Robust
decoding of h.264 encoded video transmitted over wireless


[SK95] H. Sun and W. Kwok. Concealment of damaged blocks transform


[WZ98] Y. Wang and Q. Zhu. Error control and concealment for video