Thesis for the Degree of Licentiate of Engineering

Topics in applied mathematics—error-correcting codes and homogenization of PDE

Erik Alapää

CHALMERS | GÖTEBORG UNIVERSITY

Department of Mathematics
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
Göteborg, October 2004

Topics in applied mathematics—error-correcting
codes and homogenization of PDE
ERIK ALAPÄÄ

ISSN 0347-2809/NO 2004:48
Department of Mathematics
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-772 1000

This is a thesis of the ECMI (European Consortium for Mathematics in Industry) post-graduate program in Industrial Mathematics at Chalmers University of Technology.

The work was supported by Ericsson Erisoft, Luleå, Sweden.

Abstract

This licentiate thesis consists of three papers. The first paper concerns homogenization of partial differential equations and is an introduction to the theory for industrial applications. Special emphasis is put on the research done by the homogenization groups at the universities in Narvik, Norway and Luleå, Sweden. The second paper concerns error-correcting codes. The paper describes the author's research on FEED (Far End Error Decoder), a convolutional decoder for mobile multimedia. The third paper describes some simulations designed to shed further light on the properties of FEED.

Sammanfattning

Denna licentiatavhandling består av tre artiklar. Den första artikeln berör homogenisering av partiella differentialekvationer och är en introduktion till teorin för industriella tillämpningar. Speciell vikt läggs på forskning gjord av homogeniseringsgrupperna på universiteten i Narvik och Luleå. Den andra artikeln berör felkorrigerande koder. Artikeln beskriver författarens forskning kring FEED (Far End Error Decoder), en faltningsavkodare för mobil multimedia. Den tredje artikeln beskriver simuleringar avsedda att ytterligare belysa FEED:s egenskaper.

Keywords: Homogenization of partial differential equations, error-correcting codes, channel coding, convolutional codes, streaming media, MPEG4, wireless IP, discrete Markov processes

AMS 2000 subject classification: 35B27, 74Q15, 74Q20, 74Q99, 94B10, 60J20, 60G35, 94A14, 68P30

Acknowledgements

I thank:

My supervisors, Prof. Jöran Bergh, Prof. Dag Lukkassen and Prof. Lars-Erik Persson.

Dr. James P. LeBlanc for helping me venture into the world of error-correcting codes and signal processing.

Research manager Stefan Håkansson and Prof. Tor Björn Minde at Ericsson Erisoft for giving me the opportunity to work with this project.

Drs. Christian Weiss and Thomas Stockhammer for giving me access to pre-publication papers on FEED and for taking the time to come from Munich to Stockholm to discuss FEED.

The NUHAG (Numerical Harmonic Analysis Group) at the University of Vienna, especially Prof. Hans G. Feichtinger, for their hospitality during my 3-month stay in Vienna.

Dr. Inge Söderkvist for taking the time to discuss some optimization questions that came up during the project.

The people of the free software/free UNIX world, especially the people behind LaTeX and the various tools surrounding this great typesetting system (this thesis was written in GNU Emacs on Linux and Solaris and typeset using LaTeX 2_ε with AMS-LaTeX extensions) and the people behind the GNU C/C++ compiler with surrounding tools.

My parents and my brothers for being there.

About ECMI

This licentiate thesis concludes a five semester ECMI (European Consortium for Mathematics in Industry) postgraduate programme. Of this time, about three fifths is spent taking courses in pure and applied mathematics, one fifth is spent teaching, and the rest is spent on the project presented in the thesis. At some time during the programme, a period of about four months should be spent at a foreign university. The thesis project should be based on an industrial problem.

Note

This thesis has been written under joint supervision of professors Jöran Bergh, Dag Lukkassen and Lars-Erik Persson, in a cooperation project between Chalmers University of Technology, Luleå University of Technology (both in Sweden) and Narvik University College (Norway).

This thesis consists of three papers. Note that the first two papers are large and written in monograph style, with individual tables of content. The papers are included in the thesis in the following order:

[PAPER I]	Homogenization of Partial Differential Equations: An Introduction to the Theory for Industrial Applications
[PAPER II]	Sequential Decoding of Convolutional Codes Adapted for Speech and Video Data Streams
[PAPER III]	FEED Encoded Data with Regressive Bit Energy

Homogenization of Partial Differential Equations: An Introduction to the Theory for Industrial Applications

Erik Alapää

Abstract

This paper is an introduction to homogenization research that has direct applications, i.e. in solving real-world engineering problems concerning inhomogeneous media and in practical materials science. Special emphasis is put on the work done by the homogenization groups at the universities in Luleå and Narvik. After a brief introduction to some basic mathematical tools used in homogenization, we move on to the subject of bounds for effective material properties. Some recent research by the Narvik-Luleå groups on the subject of reiterated homogenization of non-standard Lagrangians (applicable to iterated honeycomb materials) is also considered. Also, one of the main intentions of this paper is to provide a large set of references to guide the reader further in the subject matter.

Contents

The subject matter of this work is the area of homogenization of partial differential equations. Homogenization has its roots in the sixties and has in the last twenty–five years grown into a distinct field of mathematics. So, what is homogenization? Any reader with even a minimal background in engineering will not have failed to notice at least a few small waves from the "silent revolution" in materials science in the second half of the twentieth century—today, high-tech materials such as ceramics, kevlar fiber composites, titanium, super-polymers and others replace and often outperform classical engineering materials such as steel (though steel is by no means obsolete, better types of specialized steel reach the market every year). Note that *many of these new materials exhibit anisotropic properties and/or a fine-grained microstructure*. Now, consider the mathematical modelling of these materials—if we describe the rapidly varying material properties with equally rapidly varying functions, the numerical analysis of the materials will become difficult and sometimes even intractable. To put it simply, homogenization of partial differential equations has as its main purpose to approximate PDE:s that have rapidly varying coefficients with equivalent "homogenized" PDE:s that (for example) more easily lend themselves to numerical treatment in a computer.

In this article, we give an introduction to some parts of the theory with special emphasis on industrial applications. In particular, we introduce the reader to research done by the homogenization group at Luleå University of Technology, Sweden and Narvik University College, Norway (see e.g. [Luk01, BL00, JL01, LM00, Luk99, Luk97, Wal98, Luk96, Sim02, Bys02], and also section 5).

1 PDE:s with Rapidly Varying Coefficients

Suppose that we want to model a physical situation where the underlying material is heterogeneous—for example the (stationary) heat distribution in an inhomogeneous body represented by the set $\Omega \in \mathbb{R}^3$ or the deflection of an inhomogeneous membrane $\Omega \in \mathbb{R}^2$. These two physical situations can be described by the following PDE:

$$\text{Find } u \in W^{1,p}(\Omega) \text{ such that}$$

$$\begin{cases} \nabla \cdot (a^\epsilon(x, \nabla u)) = f & \text{on } \Omega, \\ u = g & \text{on } \partial\Omega. \end{cases}$$

In the equation above, the map $a : \Omega \times \mathbb{R}^N \to \mathbb{R}^N$ represents the heat conductivity in the case of the heat distribution problem or the stiffness of the material in the deflection problem, respectively. Similarly, f represents a heat source or a vertical force, and g represents the temperature or the tension at the boundary $\partial\Omega$. Finally, the parameter ϵ is very important in homogenization—it describes how quickly the material parameters vary, and in the search of an "equivalent" homogenized PDE, one considers a sequence $\{\epsilon\} \to 0$, see Fig. 1. The smaller ϵ gets, the finer the microstructure becomes. In the case where the material to be considered consists of one material periodically dispersed in another, we have the situation depicted in Fig. 2, which also shows that two different periodical "Y-cells" can be used to describe the same material. Except for the section on reiterated homogenization, we will consider mainly linear problems, i.e. when the differential operator takes the form

$$\nabla \cdot (a(x, \nabla u)) = \nabla \cdot (A \nabla u),$$

with A as an operator $A : \mathbb{R}^N \to \mathbb{R}^N$, i.e. at each point $x \in \Omega$, $A(x)$ is an $N \times N$-matrix. Also (again, except for the section on reiterated homogenization) we will restrict ourselves

Figure 1: The microstructure of Ω as $\epsilon \to 0$ *(figure from [Bys02])*.

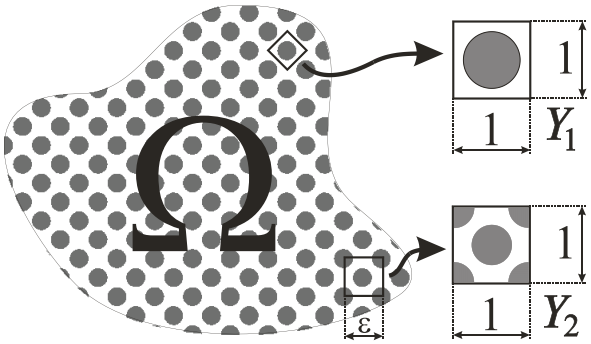

Figure 2: A periodic composite with two different Y-cells *(figure from [Bys02])*.

to the Hilbert space case, i.e. $p = 2$, $W^{1,2}(\Omega) = H^1(\Omega)$, $W_0^{1,2}(\Omega) = H_0^1(\Omega)$, where the 0 as usual denotes functions with zero trace on the boundary $\partial\Omega$.

2 Homogenization of Elliptic Partial Differential Operators

We begin with a definition (here, and in the sequel, $(\cdot,\cdot)$ denotes the scalar product of vectors in $\mathbb{R}^N$) :

Definition 1. *Let $\mathcal{O}$ denote an open set in $\mathbb{R}^N$ and let $\alpha, \beta \in \mathbb{R}$ such that $0 < \alpha < \beta$. Denote by $M(\alpha, \beta, \mathcal{O})$ the set of $N \times N$-matrices $A = (a_{ij})_{1 \le i,j \le N} \in L^\infty(\mathcal{O})^{N \times N}$ such that*

$$\begin{cases} i) & (A(x)\lambda, \lambda) \ge \alpha \, |\lambda|^2 \\ ii) & |A(x)\lambda| \le \beta \, |\lambda| \end{cases} \tag{1}$$

for any vector $\lambda \in \mathbb{R}^N$ and a.e. on $\mathcal{O}$.

This section will give a short overview of some definitions and results concerning homogenization of elliptic partial differential operators. More specifically, we will consider partial differential equations where the differential operator takes the form

$$\mathcal{A} = -\nabla \cdot (A(x) \, \nabla) = -\sum_{i,j=1}^{N} \frac{\partial}{\partial x_i} \left(a_{ij}(x) \frac{\partial}{\partial x_j} \right). \tag{2}$$

Note that if the matrix $A = I$, the identity matrix in $\mathbb{R}^N$, then the operator in (2) is the Laplacian

$$-\Delta = -\sum_{i=1}^{N} \frac{\partial^2}{\partial x_i^2}.$$

Remark 2. *Condition i) in (1) is equivalent to the classical uniform ellipticity condition for $\mathcal{A}$:*

$\exists \alpha > 0$ *such that*

$$\sum_{i,j=1}^{N} a_{ij}(x)\lambda_i\lambda_j \geq \alpha \sum_{i=1}^{N} \lambda_i^2, \qquad \text{a.e. on } \mathcal{O},\ \forall \lambda = (\lambda_1, \ldots, \lambda_N) \in \mathbb{R}^N. \quad (3)$$

In particular, the inequality above implies that $A(x)$ is invertible a.e on $\mathcal{O}$. In general, any matrix A satisfying this inequality is said to be elliptic.

If we consider A as a linear operator, $A : \mathbb{R}^N \to \mathbb{R}^N$, then by condition ii) in (1), we have

$$\|A(x)\|_2 \leq \beta \qquad \text{a.e. on } \mathcal{O},$$

where as usual, (a.e on $\mathcal{O}$) the quantity

$$\|A(x)\|_2 = \sup_{\lambda \neq 0} \frac{|A(x)\lambda|}{|\lambda|}$$

denotes the operator norm of $A(x)$, i.e. the norm of $A(x)$ as an element of $\mathcal{L}(\mathbb{R}^N, \mathbb{R}^N)$, where $\mathbb{R}^N$ is equipped with the Euclidean norm.

A vital concept in the subject of solving differential equations is of course well-posedness:

Definition 3 (Well-posedness). *Let $\mathcal{P}$ be a boundary value problem (b.v.p.) and let $\mathcal{U}$, $\mathcal{F}$ be two Banach spaces. The b.v.p. $\mathcal{P}$ is* well-posed *with respect to $\mathcal{U}$ and $\mathcal{F}$ if*

i) for any element $f \in \mathcal{F}$ there exists a solution $u \in \mathcal{U}$ of $\mathcal{F}$,

ii) the solution is unique,

iii) the map $f \in \mathcal{F} \mapsto u \in \mathcal{U}$ is continuous.

The examples we will consider here are all well-posed. They are related to the equation

$$\mathcal{A}u = -\nabla \cdot (A \nabla u) = f,$$

where $\mathcal{A}$ is given by (2) and the matrix $A \in M(\alpha, \beta, \Omega)$. Of course, a boundary value problem is formulated by supplementing this equation with boundary conditions of the appropriate type (homogeneous or nonhomogeneous Dirichlet, Neumann or Robin (i.e. mixed) conditions). Also, in homogenization another type of boundary condition is very common—if $Y = (0, l_1) \times \ldots \times (0, l_N)$ denotes a (generalized) rectangle in $\mathbb{R}^N$, then b.v.p:s with the condition $u \quad Y-$ periodic, i.e. a periodic boundary condition, are central in the branch of homogenization theory that deals with periodic structures.

2.1 The Homogeneous Dirichlet Problem

Let Ω denote an open, bounded set in $\mathbb{R}^N$ and consider the problem

$$\begin{cases} -\nabla \cdot (A\,\nabla u) = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

with the "source term" $f \in H^{-1}(\Omega)$.

By multiplying with a test function $v \in H_0^1(\Omega)$, integrating and applying Green's theorem (standard procedure in the theory of PDE:s), the corresponding variational formulation is obtained:

$$\begin{cases} \text{Find } u \in H_0^1(\Omega) \text{ such that} \\ a(u,v) = \langle f, v \rangle_{H^{-1}(\Omega), H_0^1(\Omega)}, \qquad \forall v \in H_0^1(\Omega), \end{cases} \tag{4}$$

where $a(u,v)$ is a bilinear form defined by

$$a(u,v) = \sum_{i,j=1}^{N} \int_\Omega a_{ij}(x) \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_j}\, dx = \int_\Omega A \nabla u \nabla v\, dx \qquad \forall u, v \in H_0^1(\Omega).$$

For precise conditions when the two forms of the PDE are equivalent, we refer the reader to [CD99, JKO94]. We now state a useful theorem that gives existence, uniqueness and an estimate of how sensitive problem (4) is to variations in the right-hand side (i.e. f), see [CD99]:

Theorem 4 (Homogeneous Dirichlet Problem). *Suppose that $A \in M(\alpha, \beta, \Omega)$. Then there exists a unique solution $u \in H_0^1(\Omega)$ of problem (4) for any $f \in H^{-1}(\Omega)$. Furhermore, the solution has a continuous dependence on f,*

$$\|u\|_{H_0^1(\Omega)} \leq \frac{1}{\alpha} \|f\|_{H^{-1}(\Omega)}, \tag{5}$$

where $\|u\|_{H_0^1(\Omega)} = \|\nabla u\|_{L^2(\Omega)}$.

As before, let Ω denote a bounded, open set in $\mathbb{R}^N$ and let ϵ denote a parameter which takes its values in a sequence tending to zero. Also, let

$$A^\epsilon(x) = (a_{ij}^\epsilon(x))_{1 \leq i,j \leq N} \qquad \text{a.e. on } \Omega$$

be a sequence of (non-constant) matrices such that

$$A^\epsilon \in M(\alpha, \beta, \Omega).$$

We introduce the operator

$$\mathcal{A}_\epsilon = -\nabla \cdot (A^\epsilon(x)\,\nabla) = -\sum_{i,j=1}^{N} \frac{\partial}{\partial x_i} \left(a_{ij}^\epsilon(x) \frac{\partial}{\partial x_j} \right). \tag{6}$$

Now, consider the equation

$$\mathcal{A}_\epsilon u^\epsilon = f \tag{7}$$

with a Dirichlet boundary condition on $\partial\Omega$. Equations of the type (7) model many problems in elasticity, heat conduction, electromagnetism etc. where the materials involved are

inhomogeneous with periodically varying coefficients (as before, the parameter ϵ describes the rapidly varying material properties, i.e. the heterogeneities of the material). Also, (7) is very important from a theoretical standpoint because the main mathematical obstacles in homogenization theory are already present in this basic equation.

A classical problem of type (7) is the Dirichlet problem

$$\begin{cases} -\nabla \cdot (A^\epsilon \nabla u^\epsilon) = f & \text{in } \Omega \\ u^\epsilon = 0 & \text{on } \partial\Omega, \end{cases}$$

with $f \in H^{-1}(\Omega)$, as above. For any fixed ϵ, Theorem 4 immediately gives the existence and uniqueness of a solution $u^\epsilon \in H_0^1(\Omega)$ such that

$$\int_\Omega A^\epsilon \nabla u^\epsilon \nabla v \, dx = \langle f, v \rangle_{H^{-1}(\Omega), H_0^1(\Omega)} \qquad \forall u, v \in H_0^1(\Omega).$$

Also, estimate (5) continues to hold, i.e.

$$\|u^\epsilon\|_{H_0^1(\Omega)} \leq \frac{1}{\alpha} \|f\|_{H^{-1}(\Omega)}.$$

This means that as $\epsilon \to 0$, the sequence of solutions $\{u_\epsilon\}$ is uniformly bounded. Since the space $H_0^1(\Omega)$ is a reflexive Banach space, we can apply the Eberlein-Smuljan theorem. This theorem tells us that there exists a subsequence $\{u^{\epsilon'}\}$ and an element $u^0 \in H_0^1(\Omega)$ such that

$$u^{\epsilon'} \rightharpoonup u^0 \qquad \text{weakly in } H_0^1(\Omega).$$

A few questions may now occur to the reader:

- Does u^0 satisfy some boundary problem in Ω?
- If the first question is answered in the affirmative, is u^0 uniquely determined?
- Does the "limit" boundary value problem involve a matrix A_0 that depends on the space coordinate $x \in \mathbb{R}^N$, or is A_0 a constant matrix?

It turns out that in some situations, in particular if the material properties vary periodically, it is possible to give explicit formulas for the matrix A_0 which show that A_0 is independent of the subsequence $\{u^{\epsilon'}\}$, which also implies that the solution u^0 is independent of the subsequence $\{u^{\epsilon'}\}$. As a consequence, it will follow from the Eberlein-Smuljan theorem that the whole sequence $\{u^\epsilon\}$ converges to u^0, where u^0 is the unique solution of the problem

$$\begin{cases} -\nabla \cdot (A_0 \nabla u^0) = f & \text{in } \Omega \\ u^0 = 0 & \text{on } \partial\Omega. \end{cases} \tag{8}$$

The problem (8) is called the homogenized problem, A_0 is called the homogenized matrix and u^0 the homogenized solution. To prove the statements above, powerful tools such as Tartar's method of oscillating test functions or the concept of two-scale convergence are needed. The full description of these tools lie well outside the scope of this work (For further information, we refer the interested reader to [CD99, JKO94]. Also, a useful overview of two-scale convergence is [LNW02]). However, we will give an overview of an important special case, namely homogenizaton in $\mathbb{R}^1$, where interesting results can be obtained by utilizing widely-known results from real analysis. This is the subject of the next section.

2.2 A One-Dimensional Example

Let $\Omega = (d_1, d_2)$ be an open interval in $\mathbb{R}$. Consider the problem

$$\begin{cases} -\dfrac{d}{dx}\left(a^\epsilon \dfrac{du^\epsilon}{dx}\right) = f & \text{in } (d_1, d_2) \\ u^\epsilon(d_1) = u^\epsilon(d_2) = 0. \end{cases} \tag{9}$$

The function a is assumed to be a positive function in $L^\infty(0, l_1)$ such that

$$\begin{cases} a \text{ is } l_1\text{-periodic} \\ 0 < \alpha \leq a(x) \leq \beta < \infty, \end{cases} \tag{10}$$

where α and β are constants. As usual, the notation a^ϵ from (9) means

$$a^\epsilon(x) = a(\frac{x}{\epsilon}). \tag{11}$$

The following result is classical in homogenization [Spa67]:

Theorem 5. *Let $f \in L^2(d_1, d_2)$ and a^ϵ be defined by (10) and (11). Let $u^\epsilon \in H_0^1(d_1, d_2)$ be the solution of problem (9). Then,*

$$u^\epsilon \rightharpoonup u^0 \qquad \text{weakly in } H_0^1(d_1, d_2),$$

where u^0 is the unique solution in $H_0^1(d_1, d_2)$ of the problem

$$\begin{cases} -\dfrac{d}{dx}\left(\dfrac{1}{\mathcal{M}_{(0,l_1)}(\frac{1}{a})}\dfrac{du^0}{dx}\right) = f & \text{in } (d_1, d_2) \\ u^0(d_1) = u^0(d_2) = 0. \end{cases} \tag{12}$$

(As usual, $\mathcal{M}_{(0,l_1)}(\frac{1}{a})$ denotes the mean value of $\frac{1}{a}$ over the interval $(0, l_1)$, i.e. $\mathcal{M}_{(0,l_1)}(\frac{1}{a}) = \frac{1}{|(0,l_1)|}\int_{(0,l_1)} \frac{1}{a}\, dx$).

Proof. We begin by observing that the following estimate holds:

$$\|u^\epsilon\|_{H_0^1(d_1,d_2)} \leq \frac{d_2 - d_1}{\alpha}\|f\|_{L^2(\Omega)}.$$

This inequality is a consequence of the coercivity requirement on a, the Lax-Milgram theorem and Poincaré's inequality, where $d_2 - d_1$ is the Poincaré constant. For a more detailed explanation, see [CD99]. Now, H^1 is a separable Banach space (and so is H_0^1 since it is a closed subspace of H^1), so we can invoke the Eberlein-Smuljan theorem. Thus, for a subsequence, still denoted by ϵ we have

$$\begin{cases} u^\epsilon \rightharpoonup u^0 & \text{weakly in } L^2(d_1, d_2) \\ \dfrac{du^\epsilon}{dx} \rightharpoonup \dfrac{du^0}{dx} & \text{weakly in } L^2(d_1, d_2). \end{cases} \tag{13}$$

Define

$$\xi^\epsilon = a^\epsilon \frac{du^\epsilon}{dx}$$

which satisfies

$$\frac{d\xi^\epsilon}{dx} = f \qquad \text{in } (d_1, d_2). \tag{14}$$

The estimate on u^ϵ and (10) together imply that the following estimate holds:

$$\|\xi^\epsilon\|_{L^2(d_1,d_2)} \leq \frac{\beta(d_2 - d_1)}{\alpha} \|f\|_{L^2(d_1,d_2)}.$$

Again, the Eberlein-Smuljan theorem can be invoked. We get convergence (up to a subsequence)

$$\xi^\epsilon \rightharpoonup \xi^0 \qquad \text{weakly in } L^2(d_1, d_2).$$

Furthermore, the limit ξ^0 satisfies (see [CD99])

$$\frac{d\xi^0}{dx} = f \qquad \text{in } (d_1, d_2). \tag{15}$$

The estimate on ξ^ϵ and (14) give

$$\|\xi^\epsilon\|_{L^2(d_1,d_2)} + \|\frac{d\xi^\epsilon}{dx}\|_{L^2(d_1,d_2)} \leq \frac{\beta(d_2 - d_1)}{\alpha} \|f\|_{L^2(d_1,d_2)} + \|f\|_{L^2(d_1,d_2)}.$$

Thus, ξ^ϵ is bounded in $H^1(d_1, d_2)$. The Sobolev embedding theorem then tells us that ξ^ϵ is compact in $L^2(d_1, d_2)$, and consequently there exists a subsequence, still denoted by ϵ, such that

$$\xi^\epsilon \to \xi^0 \qquad \text{strongly in } L^2(d_1, d_2).$$

The next step is to examine the relation between ξ^0 and u^0. By definition,

$$\frac{du^\epsilon}{dx} = \frac{1}{a^\epsilon} \xi^\epsilon. \tag{16}$$

The assumption on a, (10) implies that $\frac{1}{a^\epsilon}$ is bounded in $L^\infty(d_1, d_2)$, since

$$0 < \frac{1}{\beta} \leq \frac{1}{a^\epsilon} \leq \frac{1}{\alpha} < \infty. \tag{17}$$

Therefore, $\frac{1}{a^\epsilon}$ weak*-converges to the mean value of $\frac{1}{a}$, i.e.

$$\frac{1}{a^\epsilon} \rightharpoonup \mathcal{M}_{(0,l_1)}\left(\frac{1}{a}\right) = \frac{1}{l_1} \int_0^{l_1} \frac{1}{a(x)}\, dx \qquad \text{weakly}^* \text{ in } L^\infty(d_1, d_2).$$

Also, observe that due to (17),

$$\mathcal{M}_{(0,l_1)}\left(\frac{1}{a}\right) \neq 0.$$

To finish the proof, we will make use of the following theorem which is well-known in homogenization theory:

Theorem 6 (Convergence of weak-strong products). *Let E be a real Banach space and let E' denote its dual. Also, let $\{x_n\} \in E$ and $\{y_n\} \in E'$ such that*

$$\begin{cases} x_n \rightharpoonup x & \text{weakly in } E \\ y_n \to y & \text{strongly in } E'. \end{cases}$$

Then

$$\lim_{n\to\infty} \langle y_n, x_n \rangle_{E',E} = \langle y, x \rangle_{E',E}.$$

Now, since we have $\xi^\epsilon \to \xi^0$ strongly in $L^2(d_1, d_2)$ for a subsequence, the theorem above lets us pass to the limit in the weak-strong product in (16), and thus obtain

$$\frac{du^\epsilon}{dx} \rightharpoonup \mathcal{M}_{(0,l_1)}\left(\frac{1}{a}\right)\xi^0 \qquad \text{weakly in } L^2(d_1, d_2).$$

Equation (13) now gives

$$\frac{du^0}{dx} = \mathcal{M}_{(0,l_1)}\left(\frac{1}{a}\right)\xi^0.$$

Equation (15) shows that u^0 is the solution of the limit homogenized differential equation (12). Since we have shown $\mathcal{M}_{(0,l_1)}\left(\frac{1}{a}\right) \neq 0$, we know that (12) has a unique solution. Thus, we can again invoke the Eberlein-Smuljan theorem and conclude that the whole sequence $\{u^\epsilon\}$ weakly converges in $H_0^1(d_1, d_2)$ to u^0, which ends the proof. ■

3 Bounds for the Homogenized Properties

In this section, we will give a short overview of a large and important sub-area of homogenization theory—the search for bounds of the homogenized properties. For proofs of the statements in this section, see [JKO94]. We continue to consider only linear problems. More specifically, the matrix $A^\epsilon(x)$ will be of the form

$$A^\epsilon(x) = A(\frac{x}{\epsilon}), \qquad A \qquad Y - \text{periodic}.$$

Furthermore, A is assumed to be of the form $A(x) = \alpha(x)I$, where $\alpha : \mathbb{R}^N \to \mathbb{R}$ satisfies the inequalities $0 < \beta_1 \leq \alpha(x) \leq \beta_2 < \infty$. When we use the term *two-phase composite* we are talking about a material where α only takes two values, i.e. it can be written on the form

$$\alpha(x) = \alpha_1 \chi_{\Omega_1}(x) + \alpha_2 \chi_{\Omega_2}(x), \qquad \alpha_1 < \alpha_2,$$

where the sets Ω_i are the periodical extensions of $\{x \in Y : \ \alpha(x) = \alpha_i\}$ and χ_{Ω_i} denote the characteristical functions of the sets Ω_i. Also, we let m_i denote the volume fraction of the material that occupies Ω_i.

The most basic set of bounds are the *Reuss-Voigt* bounds [JKO94], which say that the homogenized matrix, denoted by b_{hom} in this section, satisfies the estimate

$$hI \leq b_{\text{hom}} \leq aI, \tag{18}$$

where I is the identity matrix and h and a denote the harmonic and arithmetic mean of α over a cell of periodicity, respectively. If A and B are matrices, then the notation $A \leq B$ means that $B - A$ has positive eigenvalues. Thus, (18) can be written

$$h \leq \lambda_i \leq a,$$

where λ_i are the eigenvalues of the homogenized matrix b_{hom}. In the particular case of a two-phase composite, the Reuss-Voigt bounds imply

$$\frac{1}{\frac{m_1}{\alpha_1}+\frac{m_2}{\alpha_2}} \le \lambda_i \le m_1\alpha_1 + m_2\alpha_2.$$

An improved set of bounds are the so-called *Hashin-Shtrikman* bounds [JKO94]. According to these,

$$L \le \frac{\lambda_1 + \cdots + \lambda_N}{N} \le U,$$

where L and U are defined by

$$L = a - \frac{\langle(\alpha-a)^2\rangle}{n\inf\alpha + \langle(\alpha-a)^2\rangle(a-\inf\alpha)^{-1}},$$
$$L = a - \frac{\langle(\alpha-a)^2\rangle}{n\sup\alpha + \langle(\alpha-a)^2\rangle(\sup\alpha-a)^{-1}}.$$

In these expressions, $\langle\cdot\rangle$ denotes the arithmetic mean. For the special case of a two-phase material, L and U take the forms

$$L = m_1\alpha_1 + m_2\alpha_2 - \frac{m_1m_2(\alpha_2-\alpha_1)^2}{n\alpha_1 + m_1(\alpha_2-\alpha_1)},$$
$$U = m_1\alpha_1 + m_2\alpha_2 - \frac{m_1m_2(\alpha_2-\alpha_1)^2}{n\alpha_2 + m_2(\alpha_1-\alpha_2)},$$

respectively. The sharpest set of bounds we will mention here are the so-called *generalized Hashin-Shtrikman* bounds [JKO94]:

$$\begin{aligned}\operatorname{tr}(b-\inf\alpha I)^{-1} &\le \frac{n\langle\frac{1}{\alpha+(n-1)\inf\alpha}\rangle}{1-\inf\alpha\langle\frac{1}{\alpha+(n-1)\inf\alpha}\rangle},\\ \operatorname{tr}(b-\sup\alpha I)^{-1} &\le \frac{n\langle\frac{1}{\alpha+(n-1)\sup\alpha}\rangle}{1-\sup\alpha\langle\frac{1}{\alpha+(n-1)\sup\alpha}\rangle},\end{aligned} \tag{19}$$

where $\operatorname{tr}(A)$ stands for the trace of the matrix A. In the particular case of a two-phase composite, (19) reduces to

$$\operatorname{tr}(b-\alpha_1 I)^{-1} = \sum_{i=1}^{n}\frac{1}{\lambda_i-\alpha_1} \le \frac{n}{m_2(\alpha_2-\alpha_1)} + \frac{m_1}{\alpha_1 m_2},$$
$$\operatorname{tr}(\alpha_2 I-b)^{-1} = \sum_{i=1}^{n}\frac{1}{\alpha_2-\lambda_i} \le \frac{n}{m_1(\alpha_2-\alpha_1)} - \frac{m_2}{\alpha_2 m_1}.$$

The research on bounds and on finding methods for obtaining bounds is a large and highly active area of investigation. As usual, various generalizations of the situation we described in this section have been, and continue to be, investigated. For example, the problem can be generalized to N phases and anisotropic behavior of the constituent materials. There are also extensions of the methods, extensions that enable the study of other types of equations such as the PDE:s that describe linear elasticity. An even more challenging problem is to find bounds for nonlinear PDE:s—for a few recent developments in this area and further references, see [Wal98].

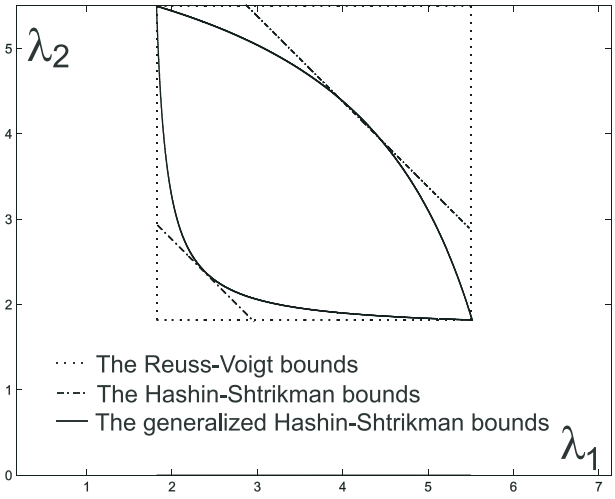

Figure 3: Gradually tighter bounds: Reuss-Voigt, Hashin-Shtrikman and generalized Hashin-Shtrikman *(figure from [Bys02])*.

4 Reiterated Homogenization

One type of material that has attracted a lot of attention in the homogenization community are the so-called *honeycomb materials*. Such a material is essentially a two-dimensonal, two-component periodic structure where the interior consists of polygons (typically rectangles or hexagons). Also, an *iterated honeycomb* is essentially a honeycomb within a honeycomb, see Figs. 4, 5, 6. The *rank* describes the "nesting level" or number of iterations.

In many areas of homogenization, the problem can be expressed in terms of minimizing an energy functional

$$F_h(u) = \int_\Omega f_h(x, \nabla u(x))\, dx,$$

where h is a scale parameter which is essentially the inverse of ϵ. If f_h can be written on the form $f_h(x,\xi) = f(hx, h^2x, \xi)$ we have a *reiterated problem* of rank 2. For more information about our group's work on reiterated homogenization, see e.g. [LLPW01]. After this short introduction to the terminology, we turn our attention to an example in the case of reiterated homogenization of non-standard Lagrangians.

4.1 Reiterated Homogenization; non-standard Lagrangians

In this section a few results concerning Γ-convergence of some reiterated non-standard Lagrangians will be presented. These Lagrangians are of the form $f(x/\epsilon, x/\epsilon^2, \xi)$ and satisfy

$$-c_0 + c_1 |\xi|^{\alpha_1} \leq f(y,z,\xi) \leq c_0 + c_2 |\xi|^{\alpha_2}. \tag{20}$$

Note the difference between (20) and a standard Lagrangian, where we would have $\alpha_1 = \alpha_2$ — here, we consider the more general case where $1 < \alpha_1 \leq \alpha_2$. The function $f(y,z,\xi)$ is assumed to be Y-periodic and Z-periodic in the first and second variables, respectively. Furthermore, f is assumed to be piecewise continuous in the first variable. Thus, f is of the

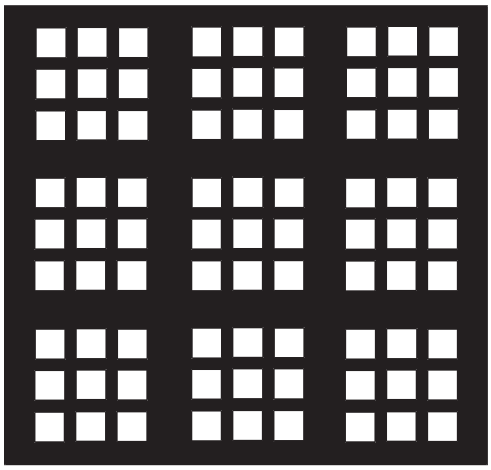

Figure 4: An iterated square honeycomb of rank 2 *(figure from [Luk99])*.

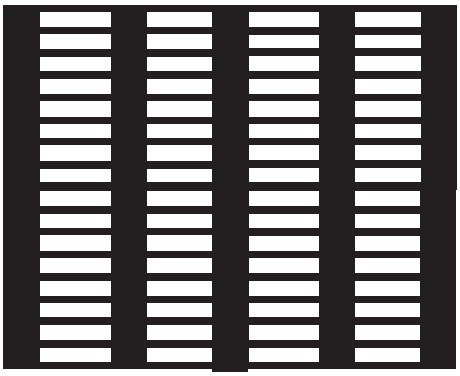

Figure 5: A laminate structure of rank 2 *(figure from [LM00])*.

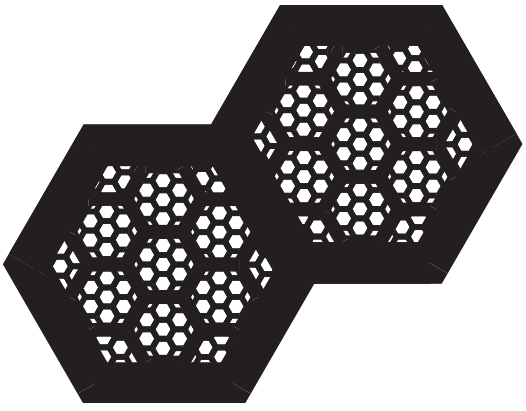

Figure 6: An iterated hexagonal honeycomb of rank 3 *(figure from [Luk97])*.

form $f(y,z,\xi) = \sum_{i=1}^{N} \chi_{\Omega_i}(y) f_i(y,z,\xi)$, where the f_i satisfy

$$\left|f_i(y,z,\xi) - f_i(y',z,\xi)\right| \leq \omega\left|y-y'\right|\left(a(z) + f_i(y,z,\xi)\right)$$

for all $y, y', z, \xi \in R^n$ and ω and a are continuous, positive, real-valued functions with $\omega(0) = 0$. In the second variable, f is assumed to be measurable, and in the third variable, f is assumed to be convex and satisfying the growth condition (20) with $1 < \alpha_1 \leq \alpha_2$.

Let $f_\epsilon(x,\xi)$ be any sequence. Assume that $f_\epsilon(x,\xi)$ is measurable in the first variable and convex and satisfying growth condition (20) in the second variable. A function $g(x,\xi)$ which satisfies the same conditions of convexity and growth as f is called the Γ_i-limit (i=1, 2) of the sequence f_ϵ if for any open, bounded set Ω with Lipschitz boundary the following conditions hold:

i) For any $u_\epsilon \in W^{1,\alpha_i}(\Omega)$ such that $u_\epsilon \rightharpoonup u$ weakly in $W^{1,\alpha_i}(\Omega)$ it holds that

$$\int_\Omega g(x,Du)\,dx \leq \liminf_{\epsilon\to 0} \int_\Omega f_\epsilon(x,Du)\,dx.$$

ii) For every $u \in W^{1,\alpha_i}(\Omega)$ there is a sequence u_ϵ such that $u_\epsilon \rightharpoonup u$ weakly in $W^{1,\alpha_i}(\Omega)$ and $u_\epsilon - u \in W_0^{1,\alpha_i}(\Omega)$,

$$\int_\Omega g(x,Du)\,dx = \lim_{\epsilon\to 0} \int_\Omega f_\epsilon(x,Du)\,dx.$$

This definition of Γ-convergence was introduced by Jikov [Jik93] and is denoted $g = \Gamma_i\text{-}\lim f_\epsilon$.

Now, let $i = \{1,2\}$, $1/\epsilon = \{1,2\}$ and put $f_\epsilon(x,\xi) = f(x/\epsilon, x/\epsilon^2, \xi)$. Again according to [Jik93] the limit $f^{\Gamma_i} = \Gamma_i - \lim f_{\epsilon_h}$ exists for some subsequence f_{ϵ_h} of f_ϵ. We now would like to mention the following result by Lukkassen [Luk01]:

Theorem 7. *It holds that $f^{\Gamma_i} = \Gamma_i - \lim f_{\epsilon_h}$ is independent of x and that*

$$Q_1^i f(\xi) \leq f^{\Gamma_i}(\xi) \leq Q_2^i f(\xi), \tag{21}$$

where

$$Q_j^i f(\xi) = \frac{1}{|Y|} \inf_{W_{per}^{1,\alpha_j}(Y)} \int_Y P^i f(y, Du(y) + \xi)\,dy$$

and

$$P^i f(y,\xi) = \frac{1}{|Z|} \inf_{W_{per}^{1,\alpha_j}(Z)} \int_Z f(y,z,Du(z)+\xi)\,dz.$$

In particular, if $P^i f$ is regular, i.e. the left and right side of (21) are equal, then the limit $f^{\Gamma_i} = \Gamma_i - \lim f_{\epsilon_h}$ exists and is given by

$$f^{\Gamma_i}(\xi) = \frac{1}{|Y|} \inf_{W_{per}^{1,t}(Y)} \int_Y P^i f(y, Du(y)+\xi)\,dy \qquad (t > 1 \text{ arbitrarily}).$$

Remark 8. *The inequalities (21) are the sharpest possible with respect to the powers α_i in $W_{per}^{1,\alpha_i}(Y)$.*

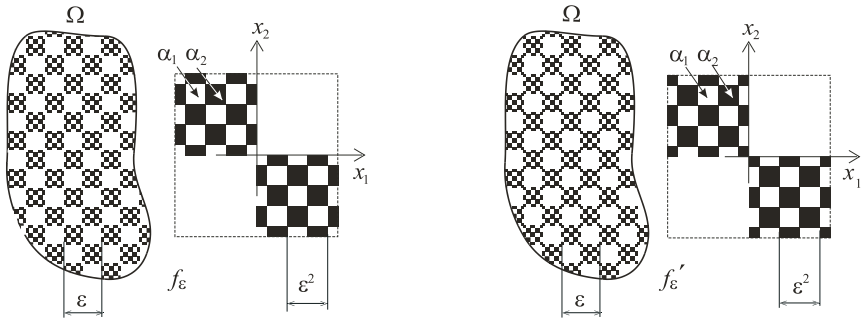

Figure 7: The functions f and f' *(figure from [Luk01]).*

An interesting question now may occur to the reader—does the (unique) limit $\Gamma - \lim f_\epsilon$ exist for any sequence $\epsilon \to 0$? The answer is affirmative if the Lagrangian function f is constant in the first variable, i.e. if f is periodic. Unlike the case when f is a standard Lagrangian (see [BL00, Luk96]), this limit does not always exist if f is dependent on the first variable. To illustrate this, we take a look at the "iterated chess Lagrangian" by Lukkassen [Luk01]:

Example 9 (Iterated chess Lagrangian). *Put $Y = Z = [-1, 1]^2$ and let χ be the characteristic function defined by*

$$\chi(x) = \begin{cases} 1 & \text{if } x_1 x_2 > 0, \\ 0 & \text{if } x_1 x_2 < 0 \end{cases}$$

on Y and then extend χ periodically to $\mathbb{R}^2$. Also, let

$$f(y, z, \xi) = \frac{1}{\alpha(y, z)} |\xi|^{\alpha(y,z)},$$

where α is defined by

$$\alpha(y, z) = \begin{cases} \alpha_1 & \text{if } \chi(y + \tau)\chi(z + \tau) = 0, \\ \alpha_2 & \text{otherwise,} \end{cases}$$

where $\tau = (1/2, 1/2)$ and $1 < \alpha_1 < 2 < \alpha_2 < \infty$. Let f' be defined as f but with a different τ, $\tau = (-1/2, 1/2)$. Now, let $f_\epsilon(x, \xi) = f(x/\epsilon, x/\epsilon^2, \xi)$ and $f'_\epsilon(x, \xi) = f'(x/\epsilon, x/\epsilon^2, \xi)$. Note that f and f' differ at 0 (see Fig. 7). If we set $i = 0$ and $1/(2\epsilon) \in \{1, 2, \ldots\}$ it can be shown that

$$\Gamma_i - \lim f_\epsilon = Q_1^i f < Q_2^i f' = \Gamma_i - \lim f'_\epsilon. \tag{22}$$

Furthermore, if we let α_1 and α_2 change place, the reverse inequality in (22) is obtained for $i = 1$. Note that in any case, $P^i f = P^i f'$. This non-regularity of $P^i f$ for the iterated chess Lagrangian is explained by the fact that $Q_1^i f(\xi)/|\xi|^{\alpha_1} \to k_1$ and $Q_2^i f(\xi)/|\xi|^{\alpha_2} \to k_2$ as $|\xi| \to \infty$ for some constants $k_1, k_2 < \infty$. The proof can be obtained using similar arguments as in p. 441 of [JKO94] (see also [JL01]).

5 Some Applications of Homogenization Theory to Materials Science

A good introduction to homogenization useful for non-mathematicians is the book [PPSW93]. One example of our cooperation with industry is the research done together with the automobile company Volvo, see e.g. [PVB01]. Other examples of our applied research are [PMW97, PBJV98, PEB02, PED03]. Here, it is also appropriate to list all the Ph.D. theses produced by the Narvik-Luleå group: [Sva92, Hol96, Luk96, Wel98, Wal98, Mei01, Bys02, Sim02]. In [Mei01] the interested reader will find further references to applied work done by our group.

References

[BL00] A. Braides and D. Lukkassen. Reiterated homogenization of integral functionals. *Math. Mod. Meth. Appl. Sci.*, 10(1):47–71, 2000.

[Bys02] Johan Byström. *Some Mathematical and Engineering Aspects of the Homogenization Theory.* PhD dissertation, Dept. of Math., Luleå University of Technology, 2002.

[CD99] Doina Cioranescu and Patrizia Donato. *An Introduction to Homogenization.* Oxford University Press, Great Clarendon Street, Oxford OX2 6DP, 1999.

[Hol96] Anders Holmbom. *Some modes of convergence and their application to homogenization and optimal composites design.* PhD dissertation, Dept. of Math., Luleå University of Technology, 1996.

[Jik93] V.V. Jikov(Zhikov). On passage to the limit in nonlinear variational problems. *Russ. Acad. Sci., Sb.*, Math 76(2):427–459, 1993.

[JKO94] V.V. Jikov(Zhikov), S.M. Kozlov, and O.A. Oleinik. *Homogenization of Differential Operators and Integral Functionals.* Springer-Verlag, Berlin-Heidelberg-New York, 1994.

[JL01] V.V. Jikov(Zhikov) and D. Lukkassen. On two types of effective conductivities. *J. Math. Anal. Appl.*, 256(1):339–343, 2001.

[LLPW01] J.-L. Lions, D. Lukkassen, L.-E. Persson, and P. Wall. Reiterated homogenization of nonlinear monotone operators. *Chin. Ann. Math., Ser. B*, 22(1):1–12, 2001.

[LM00] D. Lukkassen and G. W. Milton. On hierarchical structures and reiterated homogenization. In *Interpolation Theory and Related Topics. Proceedings of the International Conference in Honour of Jaak Peetre on his 65th Birthday*, pages 355–368, Lund, Sweden, August 17-22 2000.

[LNW02] D. Lukkassen, G. Nguetseng, and P. Wall. Two scale convergence. *Int. J. of Pure and Appl. Math.*, 2(1):35–86, 2002.

[Luk96] D. Lukkassen. *Formulae and bounds connected to optimal design and homogenization of partial differential operators and integral functionals.* PhD dissertation, Dept. of Math., Tromsø University, Norway, 1996.

[Luk97] D. Lukkassen. Bounds and homogenization of optimal reiterated honeycombs. In S. Hernández and C.A. Brebbia, editors, *Computer Aided Optimum Design of Structures V*, pages 267–276. Computational Mechanics Publications, Southampton, 1997.

[Luk99] D. Lukkassen. A new reiterated structure with optimal macroscopic behavior. *SIAM J. Appl. Math.*, 59(5):1825–1842, 1999.

[Luk01] D. Lukkassen. Reiterated homogenization of non-standard lagrangians. *C.R. Acad. Sci, Paris, Ser. I*, Math 332(11):999–1004, 2001.

[Mei01] Annette Meidell. *Homogenization and computational methods for calculating effective properties of some cellular solids and composite structures.* PhD dissertation, Dept. of Math., Norwegian University of Science and Technology (NTNU) Trondheim, 2001.

[PBJV98] Lars-Erik Persson, Johan Byström, Normund Jekabsons, and Janis Varna. Using reiterated homogenization for stiffness computation of woven composites. In David Hui, editor, *Proceedings of the International Conference on Composites Engineering, ICCE/4*, pages 133–134, Las Vegas, July 1998.

[PEB02] Lars-Erik Persson, Jonas Engström, and Johan Byström. Random versus periodic cells in homogenization. In David Hui, editor, *Proceedings of the International Conference on Composites Engineering, ICCE/9*, San Diego, July 2002.

[PED03] Lars-Erik Persson, Jonas Engström, and Johan Dasht. Degeneracy in stochastic homogenization. In David Hui, editor, *Proceedings of the International Conference on Composites Engineering, ICCE/10*, New Orleans, July 2003.

[PMW97] Lars-Erik Persson, Anette Meidell, and Peter Wall. On optimal design of two phase materials by using homogenizations. In David Hui, editor, *Proceedings of the International Conference on Composites Engineering, ICCE/4*, pages 653–654, Hawaii, July 1997.

[PPSW93] Lars-Erik Persson, Leif Persson, Nils Svanstedt, and John Wyller. *The Homogenization Method—an Introduction.* Studentlitteratur Publ., Lund, 1993.

[PVB01] Lars-Erik Persson, Niklas Bylund (Volvo), and Johan Byström. Reiterated homogenization with applications to autopart construction. In David Hui, editor, *Proceedings of the International Conference on Composites Engineering, ICCE/7*, Tenerife, July 2001.

[Sim02] Leon Simula. *Homogenization Theory for Structures of Honeycomb and Chessboard Types.* PhD dissertation, Dept. of Math., Luleå University of Technology, 2002.

[Spa67] S. Spagnolo. Sul limite delle soluzioni di problemi di cauchy relativi all'equazione del calore. *Ann. Sc. Norm. Sup.*, 21:657–699, 1967.

[Sva92] Nils Svanstedt. *G-convergence and homogenization of sequences of linear and nonlinear partial differential operators.* PhD dissertation, Dept. of Math., Luleå University of Technology, 1992.

[Wal98] Peter Wall. *Homogenization of some Partial Differential Operators and Integral Functionals.* PhD dissertation, Dept. of Math., Luleå University of Technology, 1998.

[Wel98] Niklas Wellander. *Homogenization of some linear and nonlinear partial differential equations.* PhD dissertation, Dept. of Math., Luleå University of Technology, 1998.

Sequential Decoding of Convolutional Codes Adapted for Speech and Video Data Streams

Erik Alapää

Abstract

The present work concerns the use of error-correcting codes, more specifically convolutional codes, for applications involving streaming media over wired and wireless networks. A C++ implementation of FEED, a new type of convolutional decoder especially well adapted for such applications, is described. The remainder of the work is an investigation into further improving the performance of FEED by optimizing a variable-energy modulation scheme, i.e. optimizing the discrete symbol energy distribution that is used by the actual radio transmitter in the transmission network. To provide some further support for the above investigation, a conference paper describing a series of simulations is also included.

Sammanfattning

Denna uppsats behandlar användningen av felkorrigerande koder (faltningskoder) för tillämpningar som involverar strömmande media över trådbundna och trådlösa nätverk. En C++-implementation av FEED, en ny typ av faltnings(av)kodare speciellt väl lämpad för sådana tillämpningar, beskrivs. Återstoden av arbetet utgörs av en undersökning med syfte att ytterligare förbättra FEED:s prestanda genom att optimera en modulationsmetod som utnyttjar variabel symbolenergi, d.v.s optimering av den diskreta symbolenergifördelning som används av själva radiosändardelen i transmissionssystemet. Som ytterligare stöd för ovanstående undersökning inkluderar vi också en konferensrapport som beskriver en simuleringsserie vi gjort med vår FEED-avkodare.

Contents

List of Figures

Chapter 1

Introduction

The subject area of this paper is error-correcting codes for radio transmission. The work described in the paper was part of the MulRes project that investigated transmission of multimedia signals over wired and wireless TCP/IP (Inter- and intranet) links. The industrial part of the project was to develop C++ code for FEED, a new type of sequential decoder especially well adapted for transmission of progressively (source-)encoded multimedia signals (for definition of the term progressively encoded, see section 1.2).

1.1 The Structure of this Paper

The bulk of this paper is contained in chapters 2, 3 and 4 (see below). An article describing simulations with FEED and non-uniform transmitter energy for the NORSIG 2002 conference is also included. We would also like to point out to the reader that chapter 5 contains explanations of some concepts, acronyms and terminology used in this work, and that a small index is also provided.

Chapter 2 describes the mathematics behind the FEED decoding algorithm that has roots in the 70s but was put into its current form by prof. Hagenauer's group in Munich. The chapter is intended as a coherent and easy-to-read description of material relevant to FEED from several articles by different groups (references are given in chapter 2).

Chapter 3 describes the author's C++ implementation of FEED for the Swedish company Ericsson. The chapter contains description of the simulation/test environment and key algorithms and classes in the implementation. The material should be useful to code maintainers and researchers wanting to deepen their understanding of FEED, and also for users of the FEED software.

In chapter 4 we study the question of combining FEED with a transmitter capable of applying non-uniform symbol energy to the transmitted symbols, and the results from our mathematical model indicate that a simple energy truncation should be very nearly optimal. The consequence of this is that in situations with low signal-to-noise ratio, a frame of progressively encoded information should be truncated and the saved energy/bandwidth should be used for applying more redundancy to the remaining (most important) information symbols.

Figure 1.1: Digital transmission system

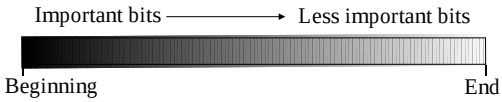

Figure 1.2: Data frame of a progressively encoded source

1.2 Background for the FEED concept

Modern source compression schemes for multimedia (sound, video etc) are often *progressive*—to explain this term, consider the transmission system depicted in Fig. 1.1 and the compression of an audio signal using some suitable transform such as the FFT or a wavelet transform. An audio frame is converted into a set of coefficients, which are sorted in decreasing size order. The ones that are below a given threshold are truncated, thus acheiving the compression. Consider the transmission of the remaining coefficients (in decreasing size order) over a noisy communication channel and reconstruction of the audio signal at the receiver end, with the following observations:

- One can reconstruct a reasonable approximation of the signal if the most important (i.e. largest) coefficients in the frame have been received correctly.
- Moving further into the frame, the reconstructed signal quality increases with the number of correctly received coefficients.
- Due to e.g. variable run-length encoding, if one coefficient has been corrupted by noise, the following coefficients in the frame can actually degrade the reconstructed quality [HSWD00].

The usual objective of channel coding (i.e. the use of error-correcting codes) is to minimize the bit error rate at the receiver, subject to constraints on transmitter power, bandwith etc. Given the observations above, another paradigm suggests itself—a channel decoder that tries to decode as much of the frame as possible (for instance, heavy noise and timing constraints could prevent the decoder from processing the whole received frame), and *delivers only the data from the beginning of the frame up to the first uncorrected error to the source decoder*. FEED is such a decoder—it is based on an algorithm that augments a sequential decoder of a convolutional code with the capability of calculating the reliability of a subpath of the decoded sequence, i.e. the probability that a given subframe of the decoded frame is error-free. Chapter 2 is devoted to explaining the mathematics behind this path reliability calculation.

Chapter 2

Theory for the FEED Decoder

The roots of the FEED concept can be traced back at least to the 70s—to two articles, [BCJR74] and [Mas72]. The term FEED itself was coined much later, by the Hagenauer group in Munich, and stands for *Far End Error Decoder* (the "far end" part will be explained in section 2.3.3). The description below is mainly based on [WSH01], but is intended for a wider audience, i.e. it contains more background material etc. for the benefit of readers who are not specialists in error-correcting codes. Besides reading the papers by FEED's inventors, [WSH01] [HSWD00], reading [BCJR74] and [Mas72] is highly recommended for anyone who wants to deepen their understanding of the FEED theory.

2.1 General Problem and Optimal Solution

We want an augmented convolutional decoder that besides estimating the transmitted sequence also computes the reliability of the estimate(s). As shown in [BCJR74] and [WSH01], this problem is a special case of a more general one: Estimating the *a-posteriori* probabilities of the states and transitions of a Markov source (see "Markov process" in chapter 5) observed through a discrete, memoryless channel.

Consider a transmission system that consists of a discrete-time finite-state Markov source observed through a DMC, followed by a decoder, as in Fig. 2.1. Fig. 2.2 shows two equivalent ways for graphical description of a time-invariant, finite state Markov source. The upper part of the figure shows a *state transition diagram* for a Markov source with three states

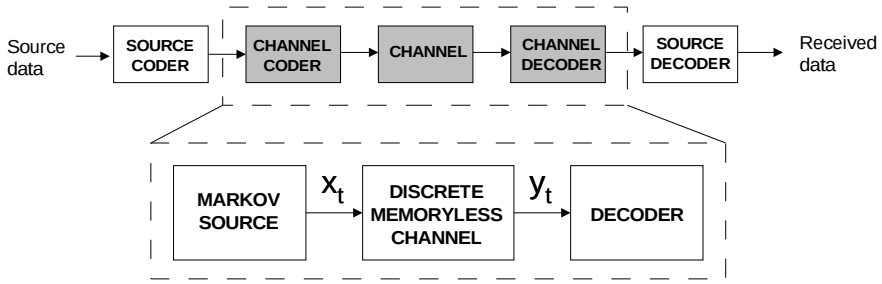

Figure 2.1: Schematic diagram of transmission system.

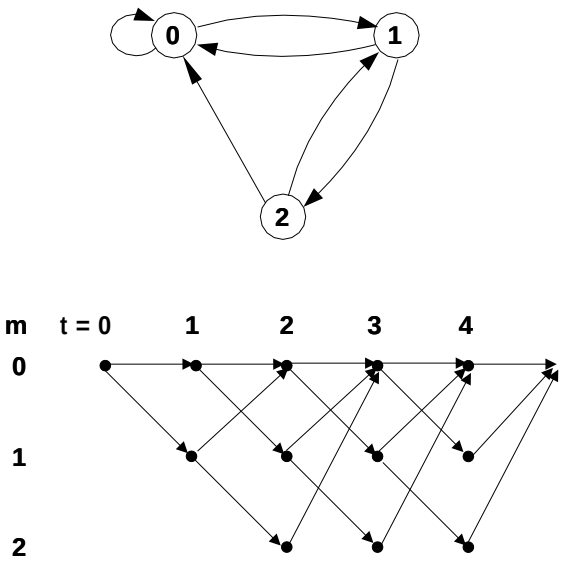

Figure 2.2: State transition diagram and equivalent trellis for a 3-state MS.

$\{0, 1, 2\}$. The arrows between the states shows the state transitions that have nonzero probability. The lower part of the figure shows a *trellis* for the same source. The horizontal axis in the trellis represents discrete time, and the vertical axis represents the three possible states. Also note that even though one single state transition diagram is insufficient to describe a time-varying Markov source, such a source could still be described by a trellis.

We now move on with the description of the system in Fig. 2.1. The Markov source has M distinct states m, $m = 0, 1, \ldots, M-1$. The state at time t is denoted by s_t and the corresponding output by x_t, where the output belongs to some finite discrete alphabet $\mathcal{X}$. The state transition probabilities $P_t(m \mid m') = \Pr\{\, s_t = m \mid s_{t-1} = m' \,\}$ are a complete description of the Markov source, and they correspond to the output distribution $Q_t(x \mid m, m') = \Pr\{\, x_t = x \mid s_{t-1} = m', s_t = m \,\}$. By convention, the Markov source always starts in the initial state $s_0 = 0$, and then generates the output sequence $\boldsymbol{x} = \boldsymbol{x}_{(0:T]} = \{x_1, x_2, \ldots, x_T\}$. The set of possible output sequences of length T is denoted by $\mathcal{S}$ and (as described above) can be represented by a trellis. Because of this, the terms path (through the trellis) and sequence will be used interchangeably. The output sequence $\boldsymbol{x}$ then travels through a DMC with transition probabilities $p_c(y_t \mid x_t)$ where the x_t belong to $\mathcal{X}$ and y_t belong to some alphabet $\mathcal{Y}$. An output sequence $\boldsymbol{y} = \boldsymbol{y}_{(0:t]} = \{y_1, y_2, \ldots, y_T\}$ is produced by the DMC, and since the DMC by definition is memoryless, we have $\Pr\{\, \boldsymbol{y}_{(0:t]} \mid \boldsymbol{x}_{(0:t]} \,\} = \prod_{\tau=1}^{t} p_c(y_\tau \mid x_\tau)$. The objective of the decoder is to produce an estimate $\hat{\boldsymbol{x}}$ of the transmitted sequence and to calculate the path reliabilities $R_{\boldsymbol{y}}(\hat{\boldsymbol{x}}_{(0:t]})$ for the decoded subpaths $\hat{\boldsymbol{x}}_{(0:t]}, 1 \le t \le T$, given the the entire received sequence $\boldsymbol{y}$. Here, $R_{\boldsymbol{y}}(\hat{\boldsymbol{x}}_{(0:t]})$ denotes the *a-posteriori* probability that, given the observation (i.e. received sequence) $\boldsymbol{y}$, the estimate $\hat{\boldsymbol{x}}_{(0:t]}$ was the transmitted sequence. Thus,

$$R_{\boldsymbol{y}}(\hat{\boldsymbol{x}}_{(0:t]}) = \Pr\{\, \hat{\boldsymbol{x}}_{(0:t]} \mid \boldsymbol{y} \,\}.$$

Set $\mathcal{S}(\boldsymbol{x}_{(a,b]}) = \{\boldsymbol{z} \in \mathcal{S} \mid \forall a < \tau \le b \quad x_\tau = z_\tau\}$, i.e. $\mathcal{S}$ denotes the set of all paths through the trellis that have a common subpath $\boldsymbol{x}_{(a,b]}, \quad 0 \le a < b \le T$, and let $s(\boldsymbol{x}_{(0:t]})$

denote the state of our Markov source after producing $\boldsymbol{x}_{(0:t]}$. For simplicity, we also set $s_0 = s(\boldsymbol{x}_{(0:0]})$. With this notation, we can write the reliability of the subpath $\hat{\boldsymbol{x}}_{(0:t]}$ given the observed sequence $\boldsymbol{y}$ as

$$R_{\boldsymbol{y}}(\hat{\boldsymbol{x}}_{(0:t]}) = \Pr\{\, \hat{\boldsymbol{x}}_{(0:t]} \mid \boldsymbol{y} \,\} = \sum_{\boldsymbol{x}' \in \mathcal{S}(\hat{\boldsymbol{x}}_{(0:t]})} \Pr\{\, \boldsymbol{x}' \mid \boldsymbol{y} \,\}.$$

Thus, $R_{\boldsymbol{y}}(\hat{\boldsymbol{x}}_{(0:t]})$ denotes the probability that the subpath $\hat{\boldsymbol{x}}_{(0:t]}$ is error-free. We then define the reliability vector

$$\boldsymbol{R}_{\boldsymbol{y}}(\hat{\boldsymbol{x}}) = \{R_{\boldsymbol{y}}(\hat{\boldsymbol{x}}_{(0:1]}), R_{\boldsymbol{y}}(\hat{\boldsymbol{x}}_{(0:2]}), \ldots, R_{\boldsymbol{y}}(\hat{\boldsymbol{x}}_{(0:T]})\},$$

consisting of the reliabilities of all subpaths of $\hat{\boldsymbol{x}}$ that begin with the first symbol of $\hat{\boldsymbol{x}}$.

We are now ready for describing the derivation of a recursive algorithm for calculating the path reliabilities. This derivation was done by the authors of [WSH01] and it is closely related to the BCJR algorithm as described in [BCJR74]. Define

$$\begin{aligned} \alpha_t(m) &= \Pr\{\, s_t = m, \boldsymbol{y}_{(0:t]} \,\} && (2.1)\\ \beta_t(m) &= \Pr\{\, \boldsymbol{y}_{(t:T]} \mid s_t = m \,\} && (2.2)\\ \gamma_t(m', m) &= \Pr\{\, s_t = m, y_t \mid s_{t-1} = m' \,\} && (2.3) \end{aligned}$$

and

$$\begin{aligned} \lambda_t(m) &= \Pr\{\, s_t = m, \boldsymbol{y}_{(0:T]} \,\}\\ \sigma_t(m', m) &= \Pr\{\, s_{t-1} = m', s_t = m, \boldsymbol{y}_{(0:T]} \,\}. \end{aligned}$$

Now,

$$\begin{aligned} \lambda_t(m) &= \Pr\{\, s_t = m, \boldsymbol{y}_{(0:t]} \,\} \cdot \Pr\{\, \boldsymbol{y}_{(t,T]} \mid s_t = m, \boldsymbol{y}_{(0:t]} \,\}\\ &= \alpha_t(m) \cdot \Pr\{\, \boldsymbol{y}_{(t,T]} \mid s_t = m \,\}\\ &= \alpha_t(m) \cdot \beta_t(m). \end{aligned}$$

Note that the middle inequality follows from the Markov property, i.e. if s_t is known, the behavior of the Markov source does not depend on $\boldsymbol{y}_{(0:t]}$. Similarly, for $\sigma_t(m', m)$ we get

$$\begin{aligned} \sigma_t(m', m) &= \Pr\{\, s_{t-1} = m', \boldsymbol{y}_{(0:t-1]} \,\} \cdot \Pr\{\, s_t = m, y_t \mid s_{t-1} = m' \,\}\\ &\quad \cdot \Pr\{\, \boldsymbol{y}_{(t,T]} \mid s_t = m \,\}\\ &= \alpha_{t-1}(m') \cdot \gamma_t(m', m) \cdot \beta_t(m). \end{aligned}$$

Thence, for $t \in \{1, 2, \ldots, T\}$

$$\begin{aligned} \alpha_t(m) &= \sum_{m'=0}^{M-1} \Pr\{\, s_{t-1} = m', s_t = m, \boldsymbol{y}_{(0:t]} \,\}\\ &= \sum_{m'} \Pr\{\, s_{t-1} = m', \boldsymbol{y}_{(0:t-1]} \,\}\\ &\quad \cdot \Pr\{\, s_t = m, y_t \mid s_{t-1} = m' \,\}\\ &= \sum_{m'} \alpha_{t-1}(m') \cdot \gamma_t(m', m), \end{aligned}$$

where the Markov property was used again to obtain the middle inequality. Since the convention is that the Markov source begins in the zero state, at $t = 0$ we also have the boundary conditions

$$\alpha_0(0) = 1 \text{ and } \alpha_0(m) = 0 \text{ for } m \neq 0.$$

Similarly, for $t \in \{1, 2, \ldots, T-1\}$

$$\begin{aligned}
\beta_t(m) &= \sum_{m'=0}^{M-1} \Pr\{s_{t+1} = m', \boldsymbol{y}_{(t,T]} \mid s_t = m\} \\
&= \sum_{m'} \Pr\{\, s_{t+1} = m', y_{t+1} \mid s_t = m \,\} \\
&\qquad \cdot \Pr\{\, \boldsymbol{y}_{(t+1,T]} \mid s_{t+1} = m' \,\} \\
&= \sum_{m'} \gamma_{t+1}(m, m') \cdot \beta_{t+1}(m'),
\end{aligned}$$

and the boundary conditions are

$$\beta_T(0) = 1 \text{ and } \beta_T(m) = 0 \text{ for } m \neq 0,$$

since the Markov source ends in the zero state by convention. For γ_t we use the description of the Markov source (the state and output transition probabilites $P_t(\cdot \mid \cdot)$ and $Q_t(\cdot \mid \cdot)$) and of the DMC (the channel transition probabilities $p_c(\cdot \mid \cdot)$) to obtain

$$\begin{aligned}
\gamma_t(m', m) &= \Pr\{\, s_t = m, y_t \mid s_{t-1} = m' \,\} \\
&= \sum_{x \in \mathcal{X}} \Pr\{\, s_t = m \mid s_{t-1} = m' \,\} \\
&\qquad \cdot \Pr\{\, x_t = x \mid s_{t-1} = m', s_t = m \,\} \cdot \Pr\{\, y_t \mid x \,\} \\
&= \sum_{x \in \mathcal{X}} P_t(m \mid m') \cdot Q_t(x \mid m', m) \cdot p_c(y_t \mid x). \qquad (2.4)
\end{aligned}$$

Equations (2.2) and (2.3) can be used to write the joint probability of $\hat{\boldsymbol{x}}_{(0:t]}$ and $\boldsymbol{y}$ as

$$\begin{aligned}
\Pr\{\, \hat{\boldsymbol{x}}_{(0:t]}, \boldsymbol{y} \,\} &= \Pr\{\, \boldsymbol{y}_{(t,T]} \mid s(\hat{\boldsymbol{x}}_{(0:t]}) \,\} \cdot \Pr\{\, \hat{\boldsymbol{x}}_{(0:t]}, \boldsymbol{y}_{(0:t]} \,\} \\
&= \beta_t(s(\hat{\boldsymbol{x}}_{(0:t]})) \prod_{\tau=1}^{t} \gamma_\tau(s(\hat{\boldsymbol{x}}_{(0:\tau]}), s(\hat{\boldsymbol{x}}_{(0:\tau-1]})).
\end{aligned}$$

To get the conditional probability

$$\Pr\{\, \hat{\boldsymbol{x}}_{(0:t]} \mid \boldsymbol{y} \,\} = \Pr\{\, \hat{\boldsymbol{x}}_{(0:t]}, \boldsymbol{y} \,\} / \Pr\{\, \boldsymbol{y} \,\}$$

we observe that since the Markov source starts in the zero state $s_0 = 0$, $\Pr\{\, \boldsymbol{y} \,\} = \sum_{m=0}^{M-1} \Pr\{\, \boldsymbol{y}_{(0:T]}, s_0 = m \,\} = \beta_0(s_0 = 0)$. Thus,

$$\begin{aligned}
R_{\boldsymbol{y}}(\hat{\boldsymbol{x}}_{(0:t]}) &= \Pr\{\, \hat{\boldsymbol{x}}_{(0:t]} \mid \boldsymbol{y} \,\} \\
&= \Pr\{\, \hat{\boldsymbol{x}}_{(0:t]}, \boldsymbol{y} \,\} / \Pr\{\, \boldsymbol{y} \,\} \\
&= \Pr\{\, \hat{\boldsymbol{x}}_{(0:t]}, \boldsymbol{y} \,\} / \beta_0(0) \\
&= \frac{\beta_t(s(\hat{\boldsymbol{x}}_{(0:t]})) \prod_{\tau=1}^{t} \gamma_\tau(s(\hat{\boldsymbol{x}}_{(0:\tau]}), s(\hat{\boldsymbol{x}}_{(0:\tau-1]}))}{\beta_0(0)}. \qquad (2.5)
\end{aligned}$$

Next, we apply (2.5) to binary convolutional codes.

2.2 Application of the General Algorithm: Path Reliability for Binary Convolutional Codes

Consider a binary convolutional encoder of rate k_0/n_0 and overall constraint length $k_0\nu$. The encoder can be implemented by k_0 shift registers, each of length ν bits, and the state of the encoder is simply the contents of the shift registers (there are other definitions of constraint length, but the definitions usually share the property that constraint length is roughly proportional to the memory of the decoder, e.g. shift register depth or word length).

For example, in the FEED implementation described in chapter 3, we use an encoder where k_0 is fixed to 1 and n_0 is usually 2 or 7 (n_0 can vary in integer steps, but the usual procedure to vary the rate is to use puncturing, which is also avaliable in our implementation). The encoder is built entirely in software and it uses 96-bit words for its operation, so the memory is 95 bits, since one bit is needed for the "current" input bit. Thus, in our FEED encoder, we have $k_0\nu = \nu = 96$.

The input to the encoder at time t is the block (i.e. symbol) of k_0 bits

$$u_t = [u_t^{(1)}, u_t^{(2)}, \ldots, u_t^{(k_0)}],$$

and the corresponding output is a block (symbol) of n_0 bits

$$x_t = [x_t^{(1)}, x_t^{(2)}, \ldots, x_t^{(n_0)}].$$

The encoder state s_t kan then be represented as the $k_0\nu$-tuple

$$s_t = [s_t^1, s_t^2, \ldots, s_t^{k_0\nu}] = [u_t, u_{t-1}, \ldots, u_{t-\nu+1}].$$

As stated before, the convention is that the encoder starts in the state $s_0 = 0$ (where the 0 really represents $k_0\nu$ zeros, but no confusion should arise from this). As an example, the default frame length in our FEED implementation is 1152 information bits and $k_0 = 1$ as stated above. This means that each information symbol u_t consists of only one bit, so to encode a frame, an information sequence $\boldsymbol{u}_{(0:T]} = [u_1, u_2, \ldots, u_T]$ with $T = 1152$ information bits would then be entered into the encoder. Again by convention, of these 1152 bits, the last $k_0\nu = 1 \cdot 96$ bits would be zeros, to ensure that the encoder returns to the zero state after encoding the frame. We will only consider time-invariant convolutional encoders, i.e. the encoder polynomials do not change during the encoding process. Such a convolutional encoder can be viewed as a finite state machine, i.e. a discrete-time, finite state, time-invariant Markov process. Thus, the encoder can be analyzed as above and represented by a trellis or, since the encoder is time-invariant, as a state transition diagram. The transition probabilities $P_t(m \mid m')$ of the trellis are governed by the input statistics. Generally, the input sequences are considered equally likely for $t \leq T - k_0\nu$ (remember, the last $k_0\nu$ bits are an all-zero "tail"). Since there are 2^{k_0} possible transitions out of each state, $P_t(m \mid m') = 2^{-k_0}$ for each of these transitions. For the tail ($t > T - k_0\nu$) there is only one possible transition out of each state, and this transition of course has probability 1. The output x_t is completely governed by the transition, so for each transition, there is a 0-1 probability distribution $Q_t(x \mid m', m)$ over the symbol alphabet $\mathcal{X}$ of binary n_0-tuples. Of course, for time-invariant codes $Q_t(\cdot \mid \cdot)$ is independent of t.

To apply (2.5), the channel properties are needed to calculate $\gamma_t(m', m)$ with (2.4). For instance, consider a DMC—during development and testing, performance simulations for

our FEED implementation have been carried out using a binary-input, 256-output DMC model. The symbol transition probabilities $p_c(y_t \mid x_t)$ can be calculated as

$$p_c(y_t \mid x_t) = \prod_{j=1}^{n_0} r(y_t^{(j)} \mid x_t^{(j)}), \tag{2.6}$$

where $r(\cdot \mid \cdot)$ are the transition probabilities for the DMC and

$$y_t = [y_t^{(1)}, y_t^{(2)}, \ldots, y_t^{(n_0)}]$$

is the block received at time t. In the binary-input, 256-output DMC model, each y_t^j could be represented by an 8-bit quantity, e.g. an `unsigned char` in C/C++.

As for algorithmic complexity, the optimal algorithm described above is unfortunately prohibitively complex—both the storage requirements and computational load increases exponentially with the number of states of the Markov source, i.e. an exponential increase with encoder memory (constraint length), see [BCJR74]. Therefore, using a suboptimal algorithm with better complexity properties is in order, and such an algorithm is described below.

2.3 Efficient, Suboptimal Decoding and Reliability Calculation Using Sequential Decoding and the Fano metric

The main problem with the algorithm described above is complexity; Shannon's noisy channel coding theorem (5.2) shows that the error probability decreases exponentially with increasing constraint lengths, i.e. long encoder memory. Thus, we would like to use long constraint lengths. However, long constraint lengths do not work well with the classical maximum-likelihood Viterbi algorithm, which has a complexity (both in memory and in number of calculations) that increases exponentially with encoder memory K. The usual solution to this problem is to use suboptimal decoding strategies, i.e. sequential decoding. Analogously, the memory requirements and computational complexity for the path-reliability-determining algorithm described above increase exponentially with the number of states of the DHMS, i.e. the encoder memory. The description below (see also [WSH01]) shows how to avoid this exponential complexity increase by calculating the path reliability in conjunction with sequential decoding. The ideas in [WSH01] make use of theory developed in [Mas72]—decoding of variable-length codes and sequential decoding using the Fano metric are essentially "isomorphic" problems, i.e. solving one problem is equivalent to solving the other.

2.3.1 A Connection Between Variable-Length Codes and the Fano Metric

Consider the transmission system shown in Fig. 2.3 for a variable-length code. Let $\mathcal{U}$ be a set of M messages (information words). To each information word $\boldsymbol{u} \in \mathcal{U}$ corresponds a code word $\boldsymbol{x_u}$

$$\boldsymbol{x_u} = [x_{\boldsymbol{u},1},\ x_{\boldsymbol{u},2},\ \ldots,\ x_{\boldsymbol{u},n_{\boldsymbol{u}}}]$$

of length $n_{\boldsymbol{u}}$. The code symbols $x_{\boldsymbol{u},j},\ j = 1, \ldots, n_{\boldsymbol{u}}$ belong to some alphabet $\mathcal{X}$, e.g. if we have a rate $1/2$ binary convolutional code each bit (in this case, with one input to the encoder,

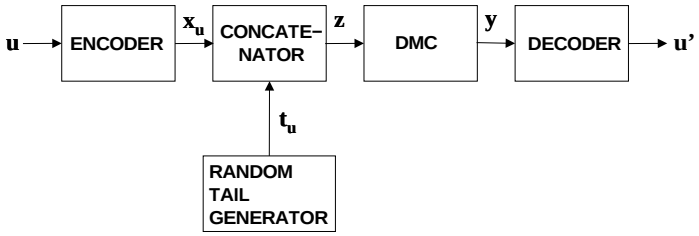

Figure 2.3: Transmission system for variable-length code

an information word symbol is one bit) in an information word $\boldsymbol{u}$ would correspond to a two-bit code word symbol $x_{\boldsymbol{u},j} \in \mathcal{X}$. Now, let $\mathcal{S}$ denote the set

$$\{\boldsymbol{x_u} \mid \boldsymbol{u} \in \mathcal{U}\}.$$

Then, $\mathcal{S}$ is our variable-length code, and we can send code words from $\mathcal{S}$ through a DMC. The goal of the variable length decoder is to estimate the transmitted variable-length codeword. The estimate is based on the received output sequence $\boldsymbol{y} = \boldsymbol{y}_{(0:T]}$, and the goal is to minimize the (*a posteriori*) error probability. Therefore, (since $\boldsymbol{y}$ is fixed, i.e. it is an observation that can be viewed as a fixed quantity for the analysis here), the estimate $\hat{\boldsymbol{u}}$ is taken as the message word $\boldsymbol{u} \in \mathcal{U}$ that maximizes $\Pr\{\, \boldsymbol{u}, \boldsymbol{y} \,\}$.

To calculate the maximizing estimate, we use the theory described in [Mas72]. Consider the abstract transmission system depicted in Fig. 2.3. Denote the maximal codeword length by T. The message $\boldsymbol{u}$ of probability $P_{\boldsymbol{u}}$ selects the codeword $\boldsymbol{x_u} = [x_{\boldsymbol{u},1}, x_{\boldsymbol{u},2}, \ldots, x_{\boldsymbol{u},n_{\boldsymbol{u}}}]$, to which is added a "random tail" $\boldsymbol{t_u} = [t_{\boldsymbol{u},1}, t_{\boldsymbol{u},2}, \ldots, t_{\boldsymbol{u},T-n_{\boldsymbol{u}}}]$, and the message plus the random tail together form a fixed-length input sequence $\boldsymbol{z} = [z_1, z_2, \ldots, z_T] = [\boldsymbol{x_u}, \boldsymbol{t_u}]$. This sequence z is then transmitted over the DMC. The random tail $\boldsymbol{t_u}$ is assumed to be selected statistically independently of $\boldsymbol{x_u}$, and we also assume that the digits in $\boldsymbol{t_u}$ are chosen independently according to a probability measure $Q()$ over the channel input alphabet, i.e,

$$\Pr\{\, \boldsymbol{t_u} \mid \boldsymbol{x_u} \,\} = \Pr\{\, \boldsymbol{t_u} \,\} = \prod_{k=1}^{T-n_{\boldsymbol{u}}} Q(t_k).$$

The introduction of the random tail $\boldsymbol{t_u}$ might confuse the reader at first, but its use will become clearer below. It suffices to think of $\boldsymbol{t_u}$ as either a convenient device for normalizing the number of received digits that must be considered in the decoding process, or as the digits resulting from subsequent encodings of further messages in a randomly selected code.

Let $\boldsymbol{y} = [y_1, y_2, \ldots, y_T]$ denote the received word. By the definition of a DMC we have

$$\Pr\{\, \boldsymbol{y} \mid \boldsymbol{z} \,\} = \prod_{t=1}^{n_{\boldsymbol{u}}} p_c(y_i \mid x_{\boldsymbol{u},i}) \prod_{j=1}^{T-n_{\boldsymbol{u}}} p_c(y_{n_{\boldsymbol{u}}+j} \mid t_j),$$

where $p_c(\cdot \mid \cdot)$ denotes the transition structure of the channel, as before.

The joint probability of sending the message $\boldsymbol{u}$, adding the random tail $\boldsymbol{t_u}$ and receiving $\boldsymbol{y}$ can then be written

$$\begin{aligned}\Pr\{\,\boldsymbol{u},\boldsymbol{t_u},\boldsymbol{y}\,\} &= P_{\boldsymbol{u}}\,\Pr\{\,\boldsymbol{t_u}\mid\boldsymbol{x_u}\,\}\,\Pr\{\,\boldsymbol{y}\mid[\boldsymbol{x_u}\boldsymbol{t_u}]\,\}\\ &= P_{\boldsymbol{u}}\prod_{i=1}^{n_{\boldsymbol{u}}} p_c(y_i\mid x_{\boldsymbol{u},i})\prod_{k=1}^{T-n_{\boldsymbol{u}}} Q(t_k)\prod_{j=1}^{T-n_{\boldsymbol{u}}} p_c(y_{n_{\boldsymbol{u}}+j}\mid t_j).\end{aligned}$$

If we replace the "dummy" index k on the last line of the previous equation by j, we get $\Pr\{\,\boldsymbol{u},\boldsymbol{t_u},\boldsymbol{y}\,\} = P_{\boldsymbol{u}}\prod_{i=1}^{n_{\boldsymbol{u}}} p_c(y_i \mid x_{\boldsymbol{u},i})\prod_{j=1}^{T-n_{\boldsymbol{u}}} p_c(y_{n_{\boldsymbol{u}}+j} \mid t_j)Q(t_j)$. If we then sum over all possible random tails and set

$$P_0(y_i) = \sum_{t_k} p_c(y_i\mid t_k)Q(t_k), \tag{2.7}$$

we get

$$\begin{aligned}\Pr\{\,\boldsymbol{u},\boldsymbol{y}\,\} &= P_{\boldsymbol{u}}\prod_{i=1}^{n_{\boldsymbol{u}}} p_c(y_i\mid x_{\boldsymbol{u},i})\prod_{j=1}^{T-n_{\boldsymbol{u}}}\left(\sum_{t_k} p_c(y_{n_{\boldsymbol{u}}+j}\mid t_k)Q(t_k)\right)\\ &= P_{\boldsymbol{u}}\prod_{i=1}^{n_{\boldsymbol{u}}} p_c(y_i\mid x_{\boldsymbol{u},i})\prod_{j=1}^{T-n_{\boldsymbol{u}}} P_0(y_{n_{\boldsymbol{u}}+j}).\end{aligned} \tag{2.8}$$

Thus, $P_0()$ is the probability measure on the channel output alphabet when the probability distribution on the channel input is $Q()$ as above. Now, given $\boldsymbol{y}$, the optimal decoding rule (optimal in the sense that the rule minimizes the probability of an erroneous decision) is to choose the message $\hat{\boldsymbol{u}}$ that maximizes $\Pr\{\,\boldsymbol{u},\boldsymbol{y}\,\}$, which is equivalent to maximizing

$$\frac{\Pr\{\,\boldsymbol{u},\boldsymbol{y}\,\}}{\prod_{i=1}^{T} P_0(y_i)},$$

since the denominator does not depend on the message $\boldsymbol{u}$. Taking logarithms, and using (2.7) and (2.8), we obtain the log-likelihood ratio

$$\begin{aligned}L(\boldsymbol{u},\boldsymbol{y}) &= \log\left[\Pr\{\,\boldsymbol{u},\boldsymbol{y}\,\}/\prod_{i=1}^{T} P_0(y_i)\right]\\ &= \log\left[P_{\boldsymbol{u}}\prod_{i=1}^{n_{\boldsymbol{u}}}\frac{p_c(y_i\mid x_{\boldsymbol{u},i})}{P_0(y_i)}\right]\\ &= \log(P_{\boldsymbol{u}}) + \sum_{i=1}^{n_{\boldsymbol{u}}}\left[\log\frac{p_c(y_i\mid x_{\boldsymbol{u},i})}{P_0(y_i)}\right]\\ &= \sum_{i=1}^{n_{\boldsymbol{u}}}\left[\log\frac{p_c(y_i\mid x_{\boldsymbol{u},i})}{P_0(y_i)} + \frac{1}{n_{\boldsymbol{u}}}\log P_{\boldsymbol{u}}\right].\end{aligned}$$

The probability $\Pr\{\,\boldsymbol{u},\boldsymbol{y}\,\}$ can equally well be written as

$$\Pr\{\,\boldsymbol{u},\boldsymbol{y}\,\} = C(\boldsymbol{y})\cdot\exp(\Lambda_{\boldsymbol{u}}) = C(\boldsymbol{y})\cdot\exp(\sum_{j=1}^{n_{\boldsymbol{u}}}\lambda_{\boldsymbol{u},j}), \tag{2.9}$$

where C is a constant that only depends on the received word $\boldsymbol{y}$, and the metric increment $\lambda_{\boldsymbol{u},j}$ for each received symbol is

$$\lambda_{\boldsymbol{u},j} = \log \frac{p_c(y_j \mid x_{\boldsymbol{u},j})}{\Pr(y_j)} + \frac{1}{n_{\boldsymbol{u}}} \log \Pr\{\, \boldsymbol{u} \,\}, \tag{2.10}$$

where $\Pr\{\, \boldsymbol{u} \,\} (= P_{\boldsymbol{u}})$ is the (*a-priori*) probability of the message $\boldsymbol{u}$ (the constant C is of course just $\prod_{i=1}^{T} P_0(y_i)$). As will be shown in section 2.3.3, in the common case of equiprobable information words, the equation (2.10) defines the well-known *Fano metric*. This metric is used for sequential decoding of convolutional codes, e.g. in the Fano algorithm. Taking logarithms does not alter the results of maximization since the logarithm is a strictly increasing function. As an additional benefit, the fact that we take logarithms gives us better numerical stability—in the original BCJR algorithm [BCJR74] and in the algorithm described in the earlier sections, we wind up with multi-factor products of probabilities, and those products quickly become small, which can cause numerical instability. Logarithms convert these products to sums, thereby avoiding having to deal with very small numbers.

2.3.2 Path Reliability for Variable-Length Codes

Given the entire received sequence $\boldsymbol{y}$ and a subpath $\boldsymbol{x}_{(0:t]}$ representing an estimate of a transmitted codeword, we want to calculate the reliability of the subpath. We begin by defining a subset $\mathcal{U}(\boldsymbol{x}_{(0:t]})$ of the information words $\mathcal{U}$ as $\mathcal{U}(\boldsymbol{x}_{(0:t]}) = \{\boldsymbol{u}' \in \mathcal{U} \mid \forall_{0<\tau\leq t}\ x_{\boldsymbol{u}',\tau} = x_\tau\}$. We then define a corresponding subset $\mathcal{S}(\boldsymbol{x}_{(0:t]})$ of the codewords $\mathcal{S}$ as $\mathcal{S}(\boldsymbol{x}_{(0:t]}) = \{\boldsymbol{x}_{\boldsymbol{u}'} \in \mathcal{S} \mid \boldsymbol{u}' \in \mathcal{U}(\boldsymbol{x}_{(0:t]})\}$. The reliability of the subpath $\boldsymbol{x}_{(0:t]}$ can now be written as

$$R_{\boldsymbol{y},\mathcal{U}}(\boldsymbol{x}_{(0:t]}) = \frac{\sum_{\boldsymbol{x}' \in \mathcal{S}(\boldsymbol{x}_{(0:t]})} \Pr(\boldsymbol{x}', \boldsymbol{y})}{\sum_{\boldsymbol{x}' \in \mathcal{S}} \Pr(\boldsymbol{x}', \boldsymbol{y})} = \frac{\sum_{\boldsymbol{u}' \in \mathcal{U}(\boldsymbol{x}_{(0:t]})} \Pr(\boldsymbol{u}', \boldsymbol{y})}{\sum_{\boldsymbol{u}' \in \mathcal{U}} \Pr(\boldsymbol{u}', \boldsymbol{y})}.$$

Using (2.9) and simplifying we get

$$\begin{aligned}
R_{\boldsymbol{y},\mathcal{U}}(\boldsymbol{x}_{(0:t]}) &= \frac{\sum_{\boldsymbol{u}' \in \mathcal{U}(\boldsymbol{x}_{(0:t]})} \Pr(\boldsymbol{u}', \boldsymbol{y})}{\sum_{\boldsymbol{u}' \in \mathcal{U}} \Pr(\boldsymbol{u}', \boldsymbol{y})} \\
&= \text{[insert (2.9)]} \\
&= \frac{\sum_{\boldsymbol{u}' \in \mathcal{U}(\boldsymbol{x}_{(0:t]})} C(\boldsymbol{y}) \cdot \exp(\sum_{j=1}^{n_{\boldsymbol{u}'}} \lambda_{\boldsymbol{u}',j})}{\sum_{\boldsymbol{u}' \in \mathcal{U}} C(\boldsymbol{y}) \cdot \exp(\sum_{j=1}^{n_{\boldsymbol{u}'}} \lambda_{\boldsymbol{u}',j})} \\
&= \text{[use the definition of } \mathcal{U}(\boldsymbol{x}_{(0:t]}) \text{ i.e. common subpath]} \\
&= \underbrace{\exp(\sum_{j=1}^{t} \lambda_{\boldsymbol{u},j})}_{\text{common subpath}} \cdot \frac{\sum_{\boldsymbol{u}' \in \mathcal{U}(\boldsymbol{x}_{(0:t]})} \exp(\sum_{j=t+1}^{n_{\boldsymbol{u}'}} \lambda_{\boldsymbol{u}',j})}{\sum_{\boldsymbol{u}' \in \mathcal{U}} \exp(\sum_{j=1}^{n_{\boldsymbol{u}'}} \lambda_{\boldsymbol{u}',j})}.
\end{aligned} \tag{2.11}$$

Equation (2.11) can be used to calculate the reliability of any decoded path $\hat{\boldsymbol{x}}_{(0:\hat{\boldsymbol{u}}]}$. If we let t vary between 1 and $n_{\hat{\boldsymbol{u}}}$, we get a vector of reliabilities of subpaths of the decoded path:

$$\boldsymbol{R}_{\boldsymbol{y},\mathcal{U}}(\hat{\boldsymbol{u}}) = \boldsymbol{R}_{\boldsymbol{y},\mathcal{U}}(\hat{\boldsymbol{x}}) = \{R_{\boldsymbol{y},\mathcal{U}}(\hat{\boldsymbol{x}}_{(0:1]}), R_{\boldsymbol{y},\mathcal{U}}(\hat{\boldsymbol{x}}_{(0:2]}), \ldots, R_{\boldsymbol{y},\mathcal{U}}(\hat{\boldsymbol{x}}_{(0:n_{\hat{\boldsymbol{u}}}]})\}.$$

2.3.3 Using the Theory for Variable-Length Codes to Calculate the Path Reliability of Sequential Decoding

If a sequential decoder, for example a Fano decoder, has to stop before it is finished (e.g. due to timing constraints), usually the partially decoded frame has to be discarded. Instead, we will make use of the partially decoded data using the theory developed in [WSH01].

The connection to the theory for variable-length codes is essentially this important observation: When the decoder has to stop before it is finished, *we have a partially explored code tree that can be viewed as a fully explored code tree for a code with variable-length codewords*! A codeword corresponding to an information word $\boldsymbol{u}$ is denoted by by $\boldsymbol{x}_{\boldsymbol{u}}$, and we have

$$\boldsymbol{x}_{\boldsymbol{u}} = \boldsymbol{x}_{\boldsymbol{u}(0:n_{\boldsymbol{u}}]} = \{(x_{\boldsymbol{u},1}^{(1)}, \ldots, x_{\boldsymbol{u},1}^{(n)}), \ldots, (x_{\boldsymbol{u},n_{\boldsymbol{u}}}^{(1)}, \ldots, x_{\boldsymbol{u},n_{\boldsymbol{u}}}^{(n)})\},$$

where n denotes the number of codeword bits per information word symbol, e.g. $(x_{\boldsymbol{u},4}^{(1)}, \ldots, x_{\boldsymbol{u},4}^{(n)})$ is the n bits in $\boldsymbol{x}_{\boldsymbol{u}}$ corresponding to the fourth symbol u_4 in the information word $\boldsymbol{u} = \boldsymbol{u}_{(0:n_{\boldsymbol{u}}]} = (u_1, \ldots, u_4, \ldots, u_{n_{\boldsymbol{u}}})$. We denote the set of information words corresponding to the partially explored code tree by $\mathcal{U}$. The joint probability of the received sequence $\boldsymbol{y}$ and a path $\boldsymbol{x}_{\boldsymbol{u}}$ can be written as an expression of the same form as (2.9):

$$\begin{aligned} \Pr\{\, \boldsymbol{x}_{\boldsymbol{u}}, \boldsymbol{y} \,\} &= \Pr\{\, \boldsymbol{u}, \boldsymbol{y} \,\} = C(\boldsymbol{y}) \cdot \exp(\Lambda_{\boldsymbol{u}}) \\ &= C(\boldsymbol{y}) \cdot \exp\Big[\sum_{j=1}^{n_{\boldsymbol{u}}} \lambda_t\big(s(\boldsymbol{u}_{(0:t-1]}), s(\boldsymbol{u}_{(0:t]})\big)\Big], \end{aligned}$$

where $s(\boldsymbol{u}_{(0:t]})$ is the state of the encoder after encoding the information sequence $\boldsymbol{u}_{(0:t]}$.

In the case where one information word symbol is one bit, assuming that the information bits are independent and equally likely to be zeros or ones, the *a priori* probability that the decoder followed the path $\boldsymbol{u}$ is

$$\Pr\{\, \boldsymbol{u} \,\} = 2^{-n_{\boldsymbol{u}}}. \tag{2.12}$$

Using (2.12), (2.10) becomes

$$\lambda_{\boldsymbol{u},j} = \log \frac{p_c(y_j \mid x_{\boldsymbol{u},j})}{\Pr(y_j)} - 1.$$

(Observe that the notation $\lambda_{\boldsymbol{u},j}$ means the same as $\lambda_j\big(s(\boldsymbol{u}_{(0:j-1]}), s(\boldsymbol{u}_{(0:j]})\big)$).

In the derivation above, there was one codeword symbol for each information word symbol. If we instead let $n_{\boldsymbol{u}}$ denote the number of *bits* in the codeword and do a "bitwise" derivation, where the ratio of the number of info bits to the number of code bits equals the code rate R, we would get

$$\Pr\{\, \boldsymbol{u} \,\} = 2^{-R\, n_{\boldsymbol{u}}},$$

and the k^{th} step "bit metric" would be

$$\lambda_{\boldsymbol{u},k} = \log \frac{r(y_k \mid x_{\boldsymbol{u},k})}{\Pr(y_k)} - R. \tag{2.13}$$

with $r(\cdot \mid \cdot)$ the "bit transition probability" for the channel, e.g. see (2.6). This "bit metric" is the well-known *Fano metric* for sequential decoding.

It is important to note that the theory developed so far is not restricted to a DMC—it could equally well have been developed for a fading channel. In that case, the metric increment becomes

$$\begin{aligned}\lambda_t(m,m') &= n\log(2) - \sum_{i=1}^{n}\log\left(1+\exp(-\psi_t^{(i)}x_t^{(i)})\right) \\ &\quad -\log(1+\exp(-L(u_t)\cdot u_t)),\end{aligned}$$

where, if $a_t^{(i)}$ denotes the fading amplitude and $\frac{E_s}{N_0}$ the SNR, $\psi_t^{(i)} = 4a_t^{(i)}\frac{E_s}{N_0}y_t^{(j)}$. The interested reader is referred to [WSH01] for details.

To sum up, we have finally arrived at the expression for calculating the path reliability in our convolutional decoder:

$$\begin{aligned}R_{\boldsymbol{y},\mathcal{U}}(\boldsymbol{u}_{(0:t]}) &= \overbrace{\exp\Big(\sum_{j=1}^{t}\lambda_j(s(\boldsymbol{u}_{(0:j-1]}),s(\boldsymbol{u}_{(0:j]}))\Big)}^{\text{common subpath}} \\ &\quad\cdot\frac{\sum_{\boldsymbol{u}'\in\mathcal{U}(\boldsymbol{u}_{(0:t]})}\exp\left[\sum_{j=t+1}^{n_{\boldsymbol{u}'}}\lambda_j\big(s(\boldsymbol{u}'_{(0:j-1]}),s(\boldsymbol{u}'_{(0:j]})\big)\right]}{\sum_{\boldsymbol{u}'\in\mathcal{U}}\exp\left[\sum_{j=1}^{n_{\boldsymbol{u}'}}\lambda_j\big(s(\boldsymbol{u}'_{(0:j-1]}),s(\boldsymbol{u}'_{(0:j]})\big)\right]}.\end{aligned}$$

This rather unwieldy expression can be understood better as

$$\begin{aligned}R_{\boldsymbol{y},\mathcal{U}}(\boldsymbol{u}_{(0:t]}) &= \text{weight of } \boldsymbol{u}_{(0:t]}\text{-path} \\ &\quad\cdot\frac{\sum\text{weight of all subpaths stemming from } \boldsymbol{u}_{(0:t]}\text{-path}}{\sum\text{weight of all paths in part. expl. code tree}},\end{aligned}$$

where "weight" simply means the metric of the subpath, i.e. the accumulated Fano metric.

This brings us back to our goal—a FEED-capable decoder. Suppose that the decoder receives a time-out before finishing, i.e. before reaching a leaf node of the code tree. We then have a partially explored code tree and an estimate $\hat{\boldsymbol{u}}$ of (part of) the transmitted sequence. The decoder then computes the subpath reliabilities of all subpaths of $\hat{\boldsymbol{u}}$ (that begin at the root node). When a large drop in the path reliability occurs, this is an indication that an uncorrected error exists, and FEED delivers the path estimate, up to this first uncorrected error, as its partially decoded frame. Here, we can also see the reason for the name FEED, *"Far End Error Decoder"*—the decoder attempts to decode as long towards the far end of the frame as possible, and then (given timing constraints etc.) delivers only the error-free part of the decoded frame; note that when we say "error-free" in this and similar contexts, we of course mean that the *likelihood* of error is comparatively small.

Chapter 3

Implementation of FEED; a FEED-Capable Fano Decoder

This chapter is intended to give an overview of our FEED implementation. For example, we hope that a programmer wanting to modify or extend the decoder can read this chapter before delving into the actual code—we highlight some of the more important classes and explain important aspects of the code. This approach is the one recommended in [Fow00]—large, comprehensive specifications that cover everything in a system tend to shine only in a few places and their sheer size makes it difficult to keep them up-to-date and correct. It is much better to highlight portions of the system that are important and/or complicated to understand just by looking at the code. Also, by focusing on a smaller subset of the system, the programmer can use more time and energy to improve the quality of the smaller specification, instead of diluting his efforts.

The system description here is done using version 1.4 of the OMG UML standard as described in [Fow00]. UML stands for Unified Modeling Language. UML was created by three of the most prominent experts in the OO industry (OO=object oriented analysis, design and programming), Grady Booch, Ivar Jacobson and James Rumbaugh, and it has become the de-facto standard for describing object-oriented systems.

We begin by listing a few facts about the implementation.

- Language for describing analysis, design and implementation: UML
- Implementation language: ISO/ANSI C++
- Compiler: gcc (the GNU C/C++ compiler), version 2.96. The code has been compiled and run on both the Sparc architecture (SUN Solaris as OS) and on the Intel X86 architecture (Linux as OS).
- Secondary compiler: icc (Intel C/C++ compiler for Linux), see remark below

Remark: The Intel C++ compiler was not used extensively for testing/verification, but it is often useful to try compiling software with different compilers from different manufacturers and on different hardware platforms to check standards compliance, compare compiler error messages and warnings, check endianness dependence etc.

The decoder is originally based on an efficient open source Fano decoder written in C by Phil Karn at Qualcomm. At the time of writing, Phil Karn's homepage is located at

`http://people.qualcomm.com/karn/`

Phil Karn's FEC software is located at

`http://people.qualcomm.com/karn/code/fec/`

and the his original Fano decoder can be downloaded from

`http://people.qualcomm.com/karn/code/fec/fano1.1.tar.gz`

Karn's decoder is rate 1/2 and the encoder has 31 bit memory, due to the fact that the Intel Pentium (X86) processor line has a word length of 4 bytes, i.e. 32 bits. The reason that the memory is 31 and not 32 bits is that one bit is required for holding the "current" bit, i.e. the most recent bit in the information sequence. We moved this decoder to C++, extended the memory to 95 bits (+"current" bit), added an interface to the Ericsson BC-lib (Baseline Codec library) and, most importantly, added the FEED functionality for predicting path reliabilities, so that the decoder can determine where in the frame the first uncorrected error occured.

3.1 How to Build The Decoder and Test Program

All the code except the BC-lib interface is built by invoking "`make`" on the command line. The BC-lib interface (i.e. `feedconvdec.*`) is built by invoking "`make bcwrapper`" on the command line. See also the `README` file in the `bc_src` subdirectory on how to build an appropriate BC-lib archive for compilation and testing.

3.2 Simulation and Test Environment; Command Line Syntax, List of Options etc.

Phil Karn's original open source encoder/decoder package also contains a test program, `seqtest`. This program has been modified to work with the FEED-capable decoder and report FEED performance data. The test program invokes the encoder to encode a frame (default frame length 1152 bits, including the "tail" of zeros needed to return the encoder to the zero state), calls a channel simulator (DMC, AWGN) to add noise to the frame, and then invokes the FEED-capable decoder. This process is repeated for a user-specified number of frames, decoding progress is reported continuously, and finally the test program writes a report of decoder performance (see the screen dumps below that demonstrate a typical test session).

Before showing an example of a typical test session, it is appropriate to give a description of the command line options. If we invoke `seqtest` without options from the command line (i.e. the UNIX shell) as follows, we get

```
~/feed> seqtest
Usage: seqtest [options] output_file
Option&default  meaning
-a 100          signal amplitude in units
-e 5.0          Signal Eb/N0 in dB (also sets -m)
-m 5.0          Eb/N0 in dB for metric table calc
```

```
-n 1152          Number of bits per frame
-N 10            Number of frames to simulate
-d 17            Decoder threshold (delta)
-l 10000         Decoder timeout, fwd motions per bit
-s [cur time]    seed for random number generator

-q               select quiet mode (default off)
-t               select timetest mode (default off)
```

Most of the options are self-explanatory, but some require a short description. The `-n` parameter sets the frame length, i.e. the number of information bits in a frame, including the all-zero tail. The `-d` parameter is the Fano decoder Δ threshold increment. In the section on the Fano algorithm in [LJ83], there is an enlightening discussion on how the Δ parameter affects decoder performance. The timeout parameter `-l` sets the number of forward motions per bit, i.e. the average number of repeats of the main loop in the Fano algorithm (see Fig. 3.1) per bit. In the example below, we set `l` to 200 and have a frame length of 1152 bits. This means that the decoder can at most use up $1152 \cdot 200$ cycles (i.e. repeats of the main loop) before a time-out occurs. The random seed parameter should be changed to achieve varied results, or held constant to reproduce bugs, errors etc. For avoiding output during simulation, the quiet mode can be selected. If the user wants to time the decoder, for instance to do profiling, the timetest mode should be chosen in order to run the simulation in a tight loop. Note also that the E_b/N_0 for the metric table calculation can be set independently of the channel simulation E_b/N_0. For instance, this could be used to test how sensitive the decoding process is to using metric tables calculated for an incorrect E_b/N_0.

For more information on how to build the encoder/decoder, test program and interface to BC-lib, see the `README` file in the same directory as the FEED C++ source files and section 3.1.

We conclude with an example run of the decoder.

```
~/feed> seqtest -l 200 -N 100 -e 1.5 outfile
Seed 1028243710 Amplitude 100 units, Eb/N0 = 1.5 dB metric table Eb/N0 = 1.5 dB
Frame length = 1152 bits, delta = 17, cycle limit = 200, #frames = 100, coderate
 = 1/2

Decoding errors:                                        0 (including 23 timeouts)
Average N:                                              73.582448
Average reliable path len:                              898.890000
Average length of timed-out frames salvaged by FEED:    51.521739
Percentage of mis-predictions by FEED:                  0.000000
  N >=  count fraction
     1    100 1
     2    100 1
     4     99 0.99
     6     93 0.93
     8     88 0.88
    10     84 0.84
    20     66 0.66
    40     45 0.45
    60     36 0.36
    80     30 0.3
   100     27 0.27
   200     23 0.23
```

3.3 Overview of decoder operation

Since the FEED decoder is based on an existing Fano algorithm convolutional decoder written in C, the decoder design is not purely object-oriented. The central part of the decoder is just a function `fano(...)` which calls methods in the various helper classes that implement the FEED functionality.

Let us begin by looking at the interface to the `fano()` function:

```
fano(
  unsigned long* metric,      // Final path metric (returned value)
  unsigned long* cycles,      // Cycle count (returned value)
  unsigned char* data,        // Decoded output data
  unsigned char* symbols,     // Raw deinterleaved input symbols
  unsigned int nbits,         // Length of uncoded bit string
  int metTab[2][256],         // Metric table, [sent sym][rx symbol]
  int delta,                  // Threshold adjust parameter
  unsigned long maxCycles,    // Decoding timeout in cycles per bit
  int* rlbPathLen)            // FEED info: length of reliable decoded
                              // path (returned value)
```

The most important of the arguments above is of course the unsigned character array `symbols`—the received data is deinterleaved and delivered to the `fano` function as the `symbols` array. Recall that the channel model we are using is a 2-input, 256-output DMC with additive white Gaussian noise. A C/C++ variable of data type `char` on most CPU architectures (including the Intel X86) is 8 bits, which is exactly what we need to represent the 256 possible output values of the DMC. The maximal value 255, i.e. hex FF, would indicate that a binary 1 was most probably transmitted, and a value of 0 would indicate a probable 0. A value close to 128 would indicate that the transmitted bit was highly distorted during transmission. For more details on the conversion of binary symbols into individual 8-bit channel symbols with specified noise and gain see the `sim.cc` module.

The `fano` function can logically be divided in two parts. The first part is the classical Fano decoding, with some added bookkeeping to be able to do the second part that does the FEED reliability calculation. For easy reference, Fig. 3.1 is a flowchart for the classical Fano algorithm. For more information on the operation of the Fano algorithm see [LJ83].

The Fano algorithm as depicted in Fig. 3.1 maps directly to the first part (i.e. the "classical Fano" part) of the decoding process, so we need only describe the second part (i.e. the FEED part) and the bookkeeping of FEED data added to the first part.

To understand the bookkeeping and the FEED implementation, assume that the decoder has received a time-out before reaching a leaf node of the code tree and consider the partially explored code tree as depicted in Fig. 3.2. If the Fano decoder's "best guess", i.e. its current path estimate when the timeout occurs, is the path leading to node B in the tree, the FEED algorithm will be used to calculate the subpath reliabilities for all subpaths of the B-path that begin at the root of the tree. For example, if we use the notation from section 2.3.3 and assume that the path estimate $\boldsymbol{u}_{(0:t]}$ corresponds to the subpath leading to from the root to

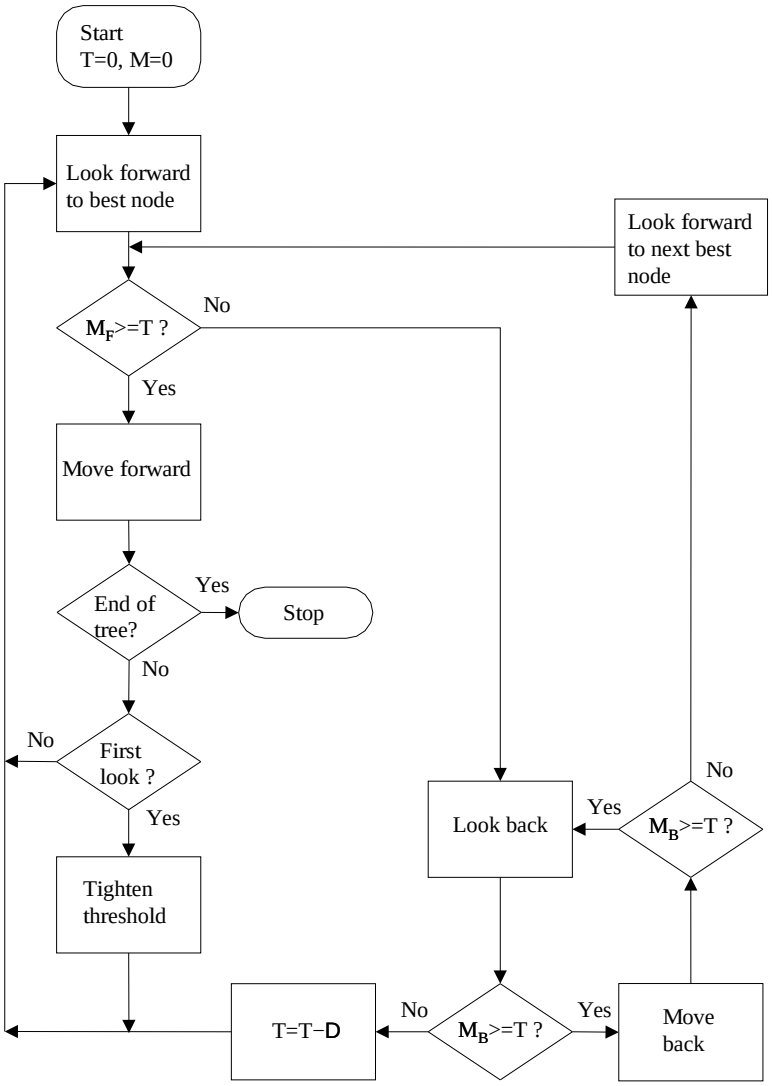

Figure 3.1: Flowchart of the Fano algorithm

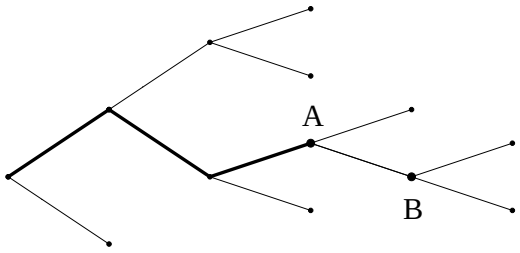

Figure 3.2: Partially explored code tree

node A in Fig 3.2, the path reliability can be calculated as

$$R_{\boldsymbol{y},\mathcal{U}}(\boldsymbol{u}_{(0:t]}) = \overbrace{\exp\Big(\sum_{j=1}^{t}\lambda_j\big(s(\boldsymbol{u}_{(0:j-1]}),s(\boldsymbol{u}_{(0:j]})\big)\Big)}^{\text{common subpath}} \cdot \frac{\sum_{\boldsymbol{u}'\in\mathcal{U}(\boldsymbol{u}_{(0:t]})}\exp\Big[\sum_{j=t+1}^{n_{\boldsymbol{u}'}}\lambda_j\big(s(\boldsymbol{u}'_{(0:j-1]}),s(\boldsymbol{u}'_{(0:j]})\big)\Big]}{\sum_{\boldsymbol{u}'\in\mathcal{U}}\exp\Big[\sum_{j=1}^{n_{\boldsymbol{u}'}}\lambda_j\big(s(\boldsymbol{u}'_{(0:j-1]}),s(\boldsymbol{u}'_{(0:j]})\big)\Big]}.$$

Recall that this rather unwieldy expression can be understood better as

$$R(A) = \text{weight of } A\text{-path} \cdot \frac{\sum \text{weight of all subpaths stemming from } A}{\sum \text{weight of all paths in part. expl. code tree}},$$

where "weight" simply means the metric of the subpath, i.e. the accumulated Fano metric.

Consequently, to be able to do the FEED path reliability calculation, the decoder needs to keep track of the partially explored code tree. Since one of the benefits of the Fano algorithm, compared to the Stack algorithm (se [LJ83]) is that only the current path estimate, and not the entire partially explored code tree, needs to be stored, this means that implementing the FEED functionality will somewhat "pollute" the basic Fano decoder. However, the Fano algorithm has other advantages over the Stack algorithm, e.g. no stack reordering has to be done. Also, the availability of a highly efficient, debugged Fano decoder implemented in C strenghtened our decision to go with the Fano algorithm.

Thus, the Fano part of the decoder needed to be augmented with some "bookkeeping", i.e. the decoder needs to continuously store and update the partially explored code tree as it loops through the Fano decoding cycle. To accomplish this, we use two data structures: `cT` is an instance of the C++ class `CodeTree` that keeps track of the partially explored code tree, and `cA` is an instance of the class `CodeArr` that holds the current path estimate. While reading this, the reader can refer to section 3.5 for UML class diagrams that describe our `CodeArr` and `CodeTree` classes, the services they provide, class hierarchies etc.

The decoder keeps working its way through the code tree until one of two things occur:

1. The decoder reaches a leaf node. The decoding is complete and the decoder delivers the contents of the `CodeArr` object `cA` as its estimate of the transmitted sequence.

2. The decoder receives a time-out before reaching a leaf node of the code tree. The FEED functionality then has to be invoked to recover parts of the timed-out frame. This is described in section 3.3.1 below.

3.3.1 Part Two of the Decoding - the Actual FEED Calculation

When the proper Fano decoding part of the `fano()` function is finished (either because of a time-out or after complete decoding), we enter the second part of the FEED decoding process. We describe this second part by looking at excerpts of the code in the `fano()` function, adding comments and explanations as we go. Note that algorithms and classes from the ISO/ANSI C++ standard are used extensively. For more information on the C++ standard library, see [Jos99]. Before delving into the code, recall how the path reliability for the path leading from the root of the tree to node A is computed:

$$R(A) = \text{weight of } A\text{-path} \cdot \frac{\sum \text{weight of all subpaths stemming from } A}{\sum \text{weight of all paths in part. expl. code tree}}.$$

Thus, we need to compute the "weight" of the whole tree, i.e. the sum of all Fano path metrics in the entire partially explored code tree, and then, for each subpath of the path estimate $\hat{\boldsymbol{u}}$, the path metric leading to the end node of the subpath (i.e. A in Fig. 3.2), and the sum of all subpath metrics for the subpaths rooted in the end node of the subpath (i.e. rooted in A). (For implementation reasons, the path reliability is actually computed in an equivalent way that is slightly different but more efficient, more on that below). Here is the first code excerpt:

```
std::vector<double> pathRlb(cA.tail() - cA.begin() + 1);
  // array of subpath reliabilities, not calculated for tail

*metric = cA.p()->gamma; // Return final path metric

const double rlbThreshold = 0.99999; // reliability threshold

if (i >= maxCycles) // decoder timed out
{
```

The main loop variable of the Fano algorithm, `i`, is greater than or equal to `maxCycles`, which means that we have received a time-out and have a partially explored code tree, stored in the `CodeTree` object `cT` and an estimate of the transmitted sequence, stored in the `CodeArr` object `cA`. Note also that the threshold for the path reliability `rlbThreshold` is very close to 1. When a subpath reliability drops below the threshold, FEED will recognize this as an uncorrected error. We move on:

```
  std::vector<long double> allMetrics; // metrics for all of the
    // partially explored code tree

  std::vector<long double> pathMetrics; // metrics for paths in U((0, t])

  double whole; // "weight" of whole partial code tree,
    // sum( exp(allMetrics) )

  double subTree; // "weight" of subtree U((0, t]),
    // sum( exp(metrics of all paths in U(0, t)) )

  while (cA.p() > cA.tail()) cA.moveBack(); // no need to calculate path
    // reliability of the all-zero tail
```

```
cT.getAllMetrics(&allMetrics);
transform(
  allMetrics.begin(), allMetrics.end(), allMetrics.begin(), scale);
transform(
  allMetrics.begin(), allMetrics.end(), allMetrics.begin(), exp);
whole = accumulate(allMetrics.begin(), allMetrics.end(), 0.0);
```

All the path metrics are retreived from the partially explored code tree. Then, the `transform()` algorithm from the C++ standard library is used to scale the metrics (simply a heuristic to avoid arithmetic overflow, does not affect the probability calculations), and to exponentiate the metrics. The exponentiation is done since the metrics are logarithms of the actual probabilities. Finally, the standard library algorithm `accumulate()` is used to compute the "weight" of the entire partially explored code tree. Before moving on, have a look at Fig. 3.2 again—since a leaf node of a partially explored code tree contains the path metric of the path leading from the root to the leaf, the reliability $R(B)$ of the path leading to node B can be computed as

$$R(B) = \frac{\text{weight of subtree } B}{\text{weight of whole partially explored tree}},$$

where "subtree B" means the subtree that consists of all paths that begin at the root and go through node B, and the weight of this subtree can be computed by simply summing the path metrics of all its leaf nodes. Next, the reliability of the path leading to node A can be computed if we have the weight of subtree A. This weight is computed by taking the weight of subtree B and adding the path metrics of the leafs in subtree A that are not also in subtree B. If we begin at node B and work our way towards the root of the partially explored code tree, we obtain the path reliabilities of all the subpaths of the B-path. This is exactly what is done in the code below:

```
BTree::Node* prevNode = 0;
for (
  subTree = 0;
  cA.p() - cA.begin() >= 0;
  prevNode = const_cast<BTree::Node*>(cT.getCurPos()), cA.moveBack())
{
  cT.getNewSubTreeMetrics(&pathMetrics, prevNode);
  transform(
    pathMetrics.begin(), pathMetrics.end(), pathMetrics.begin(), scale);
  transform(
    pathMetrics.begin(), pathMetrics.end(), pathMetrics.begin(), exp);
  subTree = accumulate(pathMetrics.begin(), pathMetrics.end(),
                       subTree);
  pathRlb[cA.p() - cA.begin()] = subTree/whole;

}
```

Thus, the above code obtains all the subpath reliabilities. All that is left to do is to loop through all the subpath reliabilities, beginning with the subpath that is only 1 symbol long, and find the longest subpath with a path reliability above the threshold. This subpath is then the sequence that FEED has salvaged from the timed-out frame. The code is straightforward and is not shown here.

3.4 Physical System Decomposition—List and Description of Files

The naming convention (.cc, .hh) for C++ files follows Ericsson (BC-lib) standard.

FILE NAME	DESCRIPTION
`bc_src/`	Used for testing only (bc-lib files)
`btree.cc`	
`btree.hh`	
`codearr.cc`	
`codearr.hh`	
`codetree.cc`	
`codetree.hh`	
`delay.cc`	
`delay.hh`	
`fano1.1.tar.gz`	Original tar-ball for the Karn Fano decoder
`fano.cc`	The actual FEED-capable Fano decoder
`fano.hh`	
`feedconvdec.cc`	See feedconvdec.hh
`feedconvdec.hh`	The BC-lib interface class
`Makefile`	See section 3.1
`metrics.cc`	
`metrics.hh`	
`README`	
`seqtest.cc`	The test program, run without args for option list
`sim.cc`	Channel simulation
`sim.hh`	
`tab.cc`	
`tab.hh`	
`uint96.cc`	Class for holding and manipulating 96-bit fields
`uint96.hh`	

3.5 Highlights of Important Classes and Their Interdependence

3.5.1 The CodeArr and CodeTree Classes

The Fano convolutional decoder uses a CodeArr object to hold the decoder's estimate of the transmitted path. The CodeArr class is used in conjunction with the CodeTree class, and the forward/backward moves of the decoder are made through the interface of this class. This way, the partially explored code tree is kept in sync with the CodeArr object. Full information hiding is not used, for two reasons; backward compatibility with the Karn Fano decoder and performance. Fig. 3.3 is an overview of the classes with their more important attributes (i.e. C++ member data) and methods(i.e. C++ member functions).

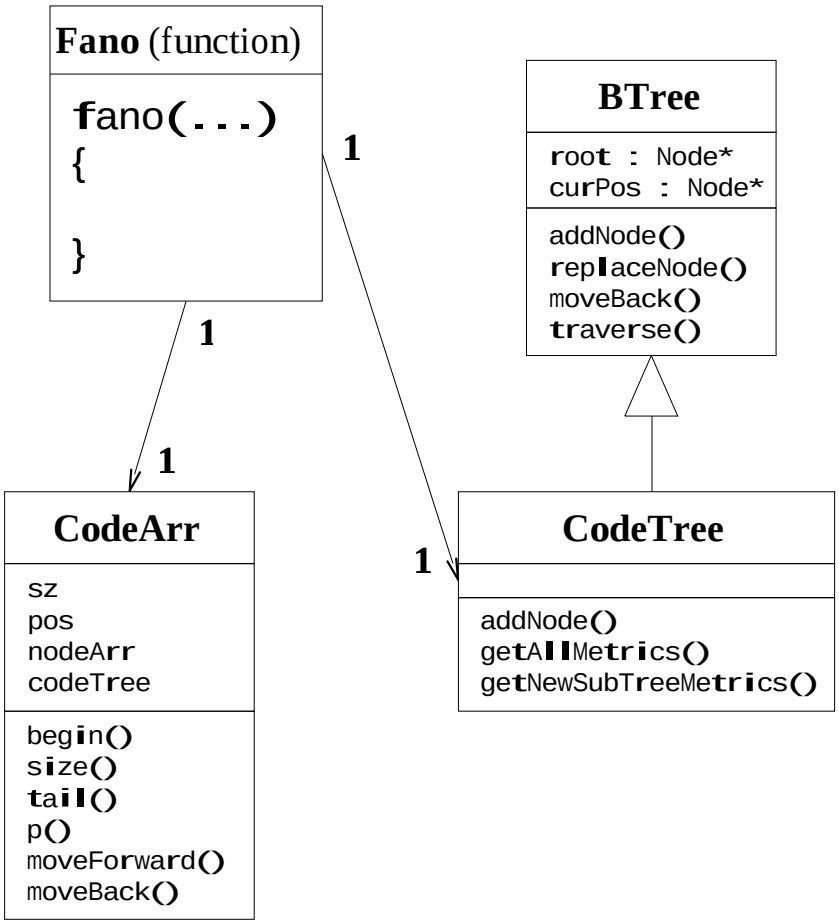

Figure 3.3: UML Class Diagram of CodeArr and CodeTree classes

Chapter 4

FEED With Optimal Non-Uniform Symbol Energy Distribution

4.1 Background

Recall that channel coding with FEED is based on a new paradigm—when transmitting data from progressively encoded sources, try to maximize the error-free length instead of minimizing the symbol error probability (note that the terms error-free length and recovered frame length are used synonymously).

In [HSWD00], puncturing is used as a means to maximize the error-free length, i.e. allocate less redundancy to the information symbols in the frame as they become less important. The physical effect of the puncturing/regressive redundancy, when looked at from a modulation/transmission viewpoint, is that less important information symbols get less total transmitter power, while more important symbols get more. Thus, an interesting alternative to puncturing suggests itself: Use a non-uniform symbol energy distribution over a frame to protect information symbols according to their importance and to maximize the error-free length. In [SL02], this idea is investigated in the case of no channel coding, i.e. given a frame of information symbols from a progressively encoded source, try to optimize the symbol energy distribution to maximize error-free length when no channel coding is applied (with no coding, each information symbol is simply one bit). Since virtually any practical system for wireless transmission benefits from channel coding, it is quite natural to ask if similar conclusions as in [SL02] hold if variable bit energy is combined with channel coding. The present chapter describes our investigations along these lines. Also, the author has made simulations combining FEED with variable bit energy, and the results of those simulations are described in [AL02].

4.2 Model

4.2.1 Channel

In this chapter, the channel model will be an AWGN channel with slow fading, where the noise level is assumed to be constant over each frame. This is a channel that is easy to work with in our optimization setting and the modelling is reasonable since modern methods that make a multi-path fading channel look like an AWGN [Vit95, Bin90] exist.

4.2.2 Notation and Formulation of the Basic Model Problem

Let N be the frame length in (code) symbols. The energy assigned to (code) symbol n in the frame is denoted by $E_s(n)$ for $n = 1, \ldots, N$. In other words, E_s denotes the symbol energy distribution (over n) we want to optimize, and E_s can be represented as a vector in $\mathbb{R}^N$. As in the uncoded investigation [SL02] we then have

$$E_s(n) \geq 0, \qquad n = 1, \ldots, N \tag{4.1a}$$

$$\sum_{n=1}^{N} E_s(n) = N, \tag{4.1b}$$

where the second constraint ensures that a constant total energy is assigned to each frame.

Let p_n denote the probability of an *uncorrected* error in code symbol n. Assume for the moment that for each n $(n = 1, \ldots, N)$ we have a functional $p_n = p(E_s)$ relating the symbol energy distribution E_s to p_n; i.e. if $\mathcal{X} \subset \mathbb{R}^n$ is the space of eligible energy distributions satisfying (4.1), for any n $(n = 1, \ldots, N)$ we can write

$$p_n : \mathcal{X} \ni E_s \mapsto p_n \in [0, 1],$$

so p_n denotes both the functional and its value at E_s. This notation is somewhat unstrict but practical, and the meaning should be clear from the context. The functional dependence is actually one of the main difficulties—for the uncoded case with the memoryless channel model described in 4.2.1, p_n will only depend on $E_s(n)$, i.e. the energy assigned to the n^{th} code symbol, but with channel coding using a convolutional code with long memory, p_n will be influenced by the energy assigned to many neighboring code symbols as well. This question will be dealt with in section 4.3.2. As stated above, the symbol energy distribution over a frame, E_s, is a vector in $\mathbb{R}^N$ (so $\mathcal{X} \subset \mathbb{R}^N$), but N is so large ($N \sim 1000$) that it is sometimes helpful to (somewhat incorrectly) think of $E_s = E_s(\cdot)$ as a smooth function curve, with n on the horizontal axis.

The error-free length l is defined to be the position of the first symbol error minus one. Assuming independent symbol errors gives

$$\Pr\{\, l = n \,\} = p_{n+1} \prod_{m=1}^{n} (1 - p_m), \tag{4.2}$$

and the expected error-free length L can be written

$$\begin{aligned} L = \mathcal{E}\{l\} &= \sum_{n=0}^{N-1} n \Pr\{\, l = n \,\} + N \prod_{m=1}^{N} (1 - p_m) \\ &= \sum_{n=1}^{N-1} n \Pr\{\, l = n \,\} + N \prod_{m=1}^{N} (1 - p_m). \end{aligned} \tag{4.3}$$

Inserting (4.2) into (4.3) gives

$$L = \mathcal{E}\{l\} = \sum_{n=1}^{N-1} \left[n\, p_{n+1} \prod_{m=1}^{n} (1 - p_m) \right] + N \prod_{m=1}^{N} (1 - p_m). \tag{4.4}$$

The assumption of independent errors is somewhat problematic. A way to compensate for this deficiency is discussed in section 4.3.2.

With L denoting the function that relates the expected error-free length to the symbol energy distribution E_s, we can phrase our optimization problem as

$$\underset{E_s \in \mathcal{X}}{\text{maximize}}\, L(E_s)$$

or

$$\begin{aligned} \underset{E_s \in \mathbb{R}^n}{\text{maximize}} \quad & L(E_s) \\ \text{subject to} \quad & E_s(n) \geq 0, \qquad n = 1, \ldots, N \\ \text{and} \quad & \sum_{n=1}^{N} E_s(n) = N. \end{aligned} \tag{4.5}$$

As mentioned above, using channel coding implies that even for a memoryless channel, changing $E_s(n)$ for any n will in general affect the symbol error probability p_i not only for the code symbol at frame pos. $i = n$ but for nearby code symbols as well. We digress briefly here to point out the important distinction between channel errors and *uncorrected* channel errors. The convolutional decoder observes the channel coder, which we regard as a Markov source, through a noisy channel (the reader might want to refer to fig 2.1). Based on the observation, i.e. received sequence of code symbols, $\boldsymbol{y}$, the decoder creates an estimate $\hat{\boldsymbol{x}}$ of the transmitted sequence $\boldsymbol{x}$. The goal of the energy optimization is to use an energy profile that makes L, the expected error-free length (i.e. L is the expected position of the first symbol error in $\hat{\boldsymbol{x}}$ minus one), as large as possible. Thus, the symbol error probabilities in $\boldsymbol{y}$ are independent if the channel is memoryless, and changing the energy of one code symbol only affects the probability of error of that symbol in $\boldsymbol{y}$, but the change in energy affects the error probability of many symbols in $\hat{\boldsymbol{x}}$ if the convolutional channel coder has long memory. As a first crude approximation, we will disregard this and optimize as if p_i only depended on $E_s(n)$ for $n = i$. This crude simplification will then be refined to obtain more realistic results in section 4.3.2. As will be shown below, even the basic model (i.e. the model using the simplification) gives rise to rather unwieldy matematical formulas and expressions—a small indication of the complexity of the problem.

4.3 Optimization of the Non-Uniform Symbol Energy Distribution

4.3.1 Optimization Using the Basic Model

Recall that our objective function L represents the expected error-free length $\mathcal{E}\{l\}$. With the simplification discussed above, we can use (4.4) to write the optimization problem (4.5) as

$$\begin{aligned}
\underset{E_s \in \mathbb{R}^n}{\text{maximize}} \quad & L(E_s) = \mathcal{E}\{l\} \\
& = \sum_{n=1}^{N-1} \left[n\, p_{n+1}[E_s(n+1)] \prod_{m=1}^{n} (1 - p_m[E_s(m)]) \right] \\
& \quad + N \prod_{m=1}^{N} (1 - p_m[E_s(m)]) && \text{(4.6a)} \\
\text{subject to} \quad & E_s(n) \geq 0, \qquad n = 1, \ldots, N && \text{(4.6b)} \\
\text{and} \quad & \sum_{n=1}^{N} E_s(n) = N. && \text{(4.6c)}
\end{aligned}$$

Using a $1 \times N$ matrix $A = [1, 1, \ldots, 1]$ to rewrite the equality constraint (4.6c), and writing $p_i[E_s(i)]$ as $p[E_s(i)]$, (4.6) becomes

$$\begin{aligned}
\underset{E_s \in \mathbb{R}^n}{\text{maximize}} \quad & L(E_s) = \mathcal{E}\{l\} \\
& = \sum_{n=1}^{N-1} \left[n\, p[E_s(n+1)] \prod_{m=1}^{n} (1 - p[E_s(m)]) \right] \\
& \quad + N \prod_{m=1}^{N} (1 - p[E_s(m)]) \\
\text{subject to} \quad & E_s(n) \geq 0, \qquad n = 1, \ldots, N \\
\text{and} \quad & A\, E_s = N.
\end{aligned}$$

To do the optimization, an expression for $p[E_s(n)]$ is required. An expression for the burst error probability P_B from page 215 of [JZ99] is used,

$$P_B < 2^{-(R_0 + o(1))mc} \qquad \text{for } R < R_0, \tag{4.8}$$

where $R = b/c$ is the code rate, encoder memory is m and R_0 denotes the computational cutoff rate. For our purposes, m and b/c will be held fixed, since we are interested in how the error probability depends on the symbol energy. This dependence arises from R_0, which is governed by the SNR, which in turn is affected by the symbol energy. Note that bounds such as (4.8) demonstrate Shannon's noisy channel coding theorem (the error probability can be made arbitrarily small and decreases exponentially with constraint length, i.e. with coder memory)—thus, such bounds are not primarily intended to be used to relate error probability to SNR (i.e. to bit energy if the noise is fixed) and are in general far from tight for all SNRs. With this "disclaimer" duly noted, we proceed. For a bandlimited AWGN channel with ideal signaling and an average input energy constraint E_s, the cutoff rate R_0 is

governed by the energy as [WC95]

$$R_0(E_s) = (\log_2 e)\left[1+\frac{E_s}{2N_0}-\sqrt{1+\left(\frac{E_s}{2N_0}\right)^2}\right] + \log_2\left[\frac{1}{2}\left(1+\sqrt{1+\left(\frac{E_s}{2N_0}\right)^2}\right)\right] \text{ bit/T}, \tag{4.9}$$

where $N_0 = 2\,\sigma^2$ is the single sided noise power spectral density per two dimensions.

Assuming constant memory m and rate b/c, equation (4.8) can be rewritten as

$$\begin{aligned} P_B &< 2^{-(R_0+o(1))mc} = 2^{-R_0mc-o(1)mc} = 2^{-R_0mc}\,2^{-o(1)mc} \\ &= 2^{-o(1)mc}\,(2^{mc})^{-R_0} = C_1C_2^{-R_0} = C_1C_2^{-R_0(E_s)}, \end{aligned} \tag{4.10}$$

which, using (4.9), gives a connection between symbol energy and the probability of an uncorrected symbol error. In FEED, a memory of 95 bits, rates between $1/2$ and $1/7$ and frame lengths on the order of a thousand bits are used. For practical reasons (i.e. to reduce computational complexity, increase numerical stability etc.) a frame length of 100 bits was used in the optimization. Also, in the refined model (see below), a convolution with a window function with length (i.e. support) on the order of 10 bits was used, corresponding to a memory of around 10 bits. The memory length was chosen to be approximately $1/10$ of the frame length, as in the real problem. Of course, the constants above were also changed. Thus, our analysis will be qualitative, but there is no reason why the qualitative results would be different in the "scaled down" version of the problem. See also the conclusions, section 4.4.

Equation (4.9) holds for code rates R below the cutoff R_0. In our case, R is held fixed at $\frac{1}{2}$ and the cutoff R_0 decreases with SNR. For SNRs that are so low that R_0 becomes lower than $\frac{1}{2}$, (4.9) is replaced by an estimation based on interpolation. The interpolation emulates the actual behavior of FEED observed in simulations, i.e. if the SNR is lowered beneath a threshold, the error probability and the time for decoding a frame increases dramatically.

For the optimization, the gradient of the objective function L is needed. The gradient will involve the derivative $\frac{dP_B}{dE_s}$ which is calculated below.

$$\frac{dP_B}{dE_s} = \frac{d}{dE_s}C_1C_2^{-R_0(E_s)} = -C_1\ln C_2 R_0'(E_s)C_2^{-R_0(E_s)}, \tag{4.11}$$

with

$$
\begin{aligned}
R_0'(E_s) = \frac{d}{dE_s} \Bigg(& (\log_2 e) \left[1 + \frac{E_s}{2N_0} - \sqrt{1 + \left(\frac{E_s}{2N_0} \right)^2} \right] \\
& + \log_2 \left[\frac{1}{2} \left(1 + \sqrt{1 + \left(\frac{E_s}{2N_0} \right)^2} \right) \right] \Bigg) \\
= (\log_2 e) & \left[\frac{1}{2N_0} - \frac{1}{2\sqrt{1 + \left(\frac{E_s}{2N_0} \right)^2}} \frac{E_s}{2N_0^2} \right] \\
& + \frac{1}{\ln 2 \left[\frac{1}{2} \left(1 + \sqrt{1 + \left(\frac{E_s}{2N_0} \right)^2} \right) \right]} \frac{1}{2} \left(\frac{1}{2\sqrt{1 + \left(\frac{E_s}{2N_0} \right)^2}} \frac{E_s}{2N_0^2} \right) \\
= (\log_2 e) & \left[\frac{1}{2N_0} - \frac{E_s}{4N_0^2 \sqrt{1 + \left(\frac{E_s}{2N_0} \right)^2}} \right] \\
& + \frac{E_s}{4N_0^2 \ln 2 \left(1 + \sqrt{1 + \left(\frac{E_s}{2N_0} \right)^2} \right) \sqrt{1 + \left(\frac{E_s}{2N_0} \right)^2}}.
\end{aligned}
$$

The gradient ∇L depends on the vector of probabilites $[\, p_1, \ldots, p_N \,]$, which in turn depends on the symbol energy vector $E_s = [\, E_s(1), \ldots, E_s(N) \,]$. In the general case

$$
L = L(p_1, \ldots, p_N) = L\big(p_1(E_s(1), \ldots, E_s(N)) \ , \ \ldots \ , \ p_N(E_s(1), \ldots, E_s(N))\big),
$$

so in general, the n^{th} component of the gradient vector is

$$
\frac{\partial L}{\partial E_s(n)} = \frac{\partial L}{\partial p_1} \frac{\partial p_1}{\partial E_s(n)} + \cdots + \frac{\partial L}{\partial p_N} \frac{\partial p_N}{\partial E_s(n)}.
$$

With the basic model (i.e. with the crude initial approximation that p_n, the probability of an uncorrected error in symbol n, only depends on $E_s(n)$), the expression above simplifies to

$$
\frac{\partial L}{\partial E_s(n)} = \frac{\partial L}{\partial p_n} \frac{\partial p_n}{\partial E_s(n)}.
$$

Using (4.10) and (4.11) gives

$$
\begin{aligned}
\frac{\partial L}{\partial E_s(n)} &= \frac{\partial L}{\partial p_n} \frac{\partial p_n}{\partial E_s(n)} \\
&= \frac{\partial L}{\partial p_n} \frac{\partial}{\partial E_s(n)} C_1 C_2^{-R_0(E_s(n))} \\
&= -C_1 \ln C_2 R_0'(E_s(n)) C_2^{-R_0(E_s(n))} \frac{\partial L}{\partial p_n}.
\end{aligned}
$$

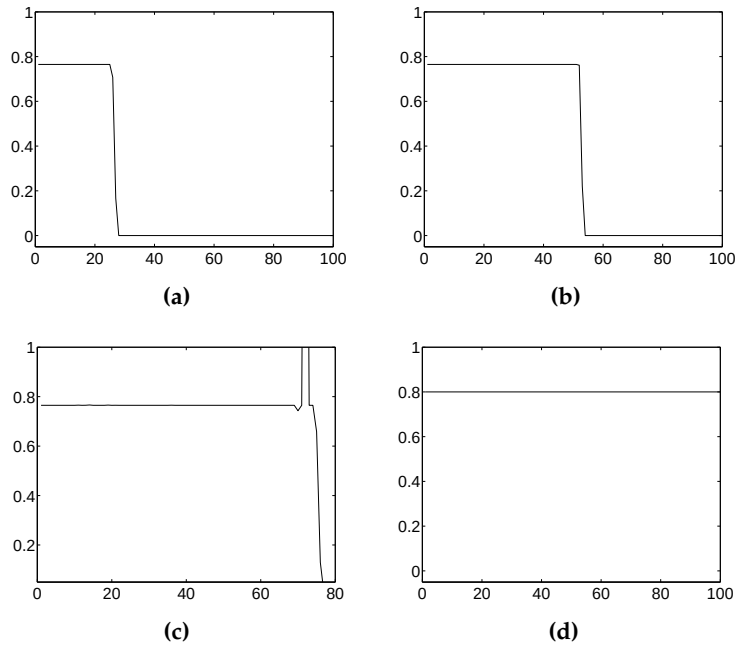

Figure 4.1: Optimal energy profiles for the basic model (i.e. without conv). Subfigure a)–d) shows symbol energy versus frame length for 20, 40, 60 and 80 percent of unit energy per symbol, respectively. The frame length is 100 symbols—normally, frame lengths in FEED are on the order of 1000 symbols, but the problem has been scaled down to avoid impractically long calculations in Matlab. Note: The narrow peak in (c) is not significant (it represents very little energy) and is a result of finite-precision arithmetic.

Furthermore,

$$\begin{aligned}\frac{\partial L}{\partial p_n} &= \frac{\partial}{\partial p_n}\left(\sum_{k=1}^{N-1}\left[k\,p_{k+1}\prod_{m=1}^{k}(1-p_m)\right]+N\prod_{m=1}^{N}(1-p_m)\right)\\ &= (n-1)\Pi_{n-1}-\sum_{k=n}^{N-1}kp_{k+1}\frac{\Pi_k}{1-p_n}-N\frac{\Pi_N}{1-p_n},\end{aligned}$$

with the compact notation $\Pi_n = \prod_{m=1}^{n}(1-p_m)$.

For the actual optimization calculations, we used the `fmincon` algorithm in Matlab. This algorithm is a very powerful and flexible general-purpose tool for constrained optimization. For more information on `fmincon`, we refer the interested reader to the Matlab documentation. Using the basic model in the optimization gives the symbol energy distributions shown in Fig. 4.1. Note that the energy distributions that maximize the objective function of the basic model can be approximated very well by a simple truncation, i.e. the optimal approach is to send only part of the frame. Further conclusions are postponed to section 4.4, where we compare with results for a more realistic model from section 4.3.2 and earlier experimental and theoretical results.

4.3.2 Optimization Using a Refined Model

It is obvious that in general, changing the energy of (code) symbol n affects the probability of an uncorrected error not only for symbol n but for other symbols in the frame as well. This also has to be taken into account when optimizing for maximum error-free length. As mentioned above, in general, the n^{th} component of ∇L is $\frac{\partial L}{\partial E_s(n)} = \sum_{k=1}^{N} \frac{\partial L}{\partial p_k} \frac{\partial p_k}{\partial E_s(n)}$. The "spreading" of the energy is modeled using a weight function $w(k)$, centered around n by translation. Assuming that $w(k)$ is even, the spreading can be written as a convolution:

$$\begin{aligned} \frac{\partial L}{\partial E_s(n)} &= \sum_{k=1}^{N} \frac{\partial L}{\partial p_k} \frac{\partial p_k}{\partial E_s(n)} \\ &\approx \sum_{k=1}^{N} \frac{\partial L}{\partial p_k} w_{k-n} \frac{\partial p_n}{\partial E_s(n)} \\ &= \frac{\partial p_n}{\partial E_s(n)} \sum_{k=1}^{N} \frac{\partial L}{\partial p_k} w_{n-k} \\ &= \frac{\partial p_n}{\partial E_s(n)} \left(\frac{\partial L}{\partial p} * w \right)(n), \end{aligned} \tag{4.12}$$

where $\frac{\partial L}{\partial p} = [\frac{\partial L}{\partial p_1}, \ldots, \frac{\partial L}{\partial p_N}]$, and $w(k) = w_k$ is used for notational coherence. The reader may also refer to section 4.5 for a few practical remarks concerning convolution and its use in this context.

Here, it is appropriate to elaborate a bit on the modelling. The fundamental idea in error-correcting codes is to add redundancy in a controlled fashion to protect against channel errors. Since the encoding process in a convolutional encoder achieves this by convolving the input symbols with earlier symbols, it is quite natural to model this in our optimization by "spreading" or "smoothing" the error probabilities by convolving with a weight function. Since the exact form of this weight function must be chosen somewhat arbitrarily, it is important to check if the conclusions from the optimization are sensitive to the chosen shape of the function. This has been done, and the optima remain essentially unchanged when varying the shape of the weight function.

The objective function L also needs to be modified. The same weight function w is used to obtain a smoother, more realistic vector of probabilities $p = [p_1, \ldots, p_N]$. This is accomplished by calculating the individual $p_n, \quad (n = 1, \ldots, N)$ as before and then convolving with w. If the modified probabilities are denoted $\tilde{p}$, this can be written

$$\tilde{p}_n = (p * w)(n) = \sum_k p_k w_{n-k}, \tag{4.13}$$

where the sum is over all k that give valid indices for the factors (again, refer to section 4.5). Fig. 4.2 shows an example of a Gaussian-shaped weight function, and Fig. 4.3 shows the smoothing effect of convolution.

As mentioned in the derivation of equation (4.2), the assumption of independent errors is problematic—remember that the entire discussion is based on probabilities of *uncorrected* channel errors, and for a convolutional decoder, such errors are in general not independent (if the decoder chooses the wrong path through the trellis, several symbols will be erroneous before it returns to the correct path). Thus, the derivation of the expression for $\Pr\{\, l = n \,\}$

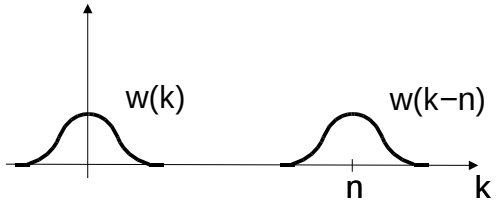

Figure 4.2: A weight function w and a translated version of the same function.

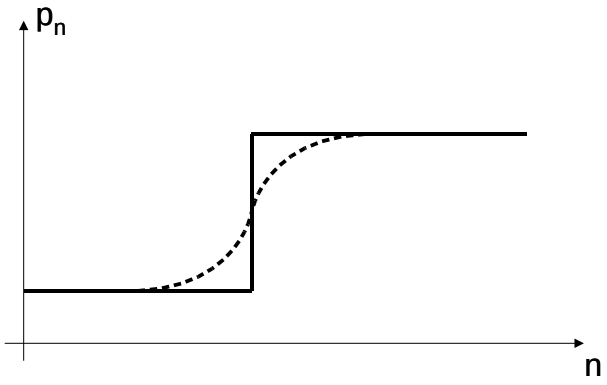

Figure 4.3: Error probabilities before and after convolution with w.

is somewhat simplified. Due to the complexity of the problem, it is difficult to modify (4.2) to account for dependent errors, but the refined model at least partly compensates for this—since the vector of error probabilities is smoothed out by convolution, the error probabilities become dependent in the sense that changing the error probability for one n also changes the probabilities for "neighboring" n.

Using the refined model in the optimization gives the optima shown in Fig. 4.4. Thus, using a more realistic model still leads to the conclusion that the optimal energy distribution can be approximated very well by a truncation.

4.3.3 Does There Exist a Global Optimum, and Have We Found It?

A very important verification is of course to check that the optima we found are indeed global optima. This was done by putting in a large set of random energy distributions as start values for the optimization, and checking that the final output remained the same. This "health check" worked out perfectly—an example of this is shown in Fig 4.5.

4.3.4 Discussion of the Shape of the Optimal Symbol Energy Distribution

[SL02] shows that in the uncoded case, the optimal symbol energy distribution is not exactly a truncation but a continuous curve that can be approximated rather accurately by a truncation (the truncation performs within a few percent of the actual optimum). Adding error-correcting coding to a digital transmission system often has the effect of sharpening thresholds, e.g. with the noise level on one side of a threshold the transmission is nearly

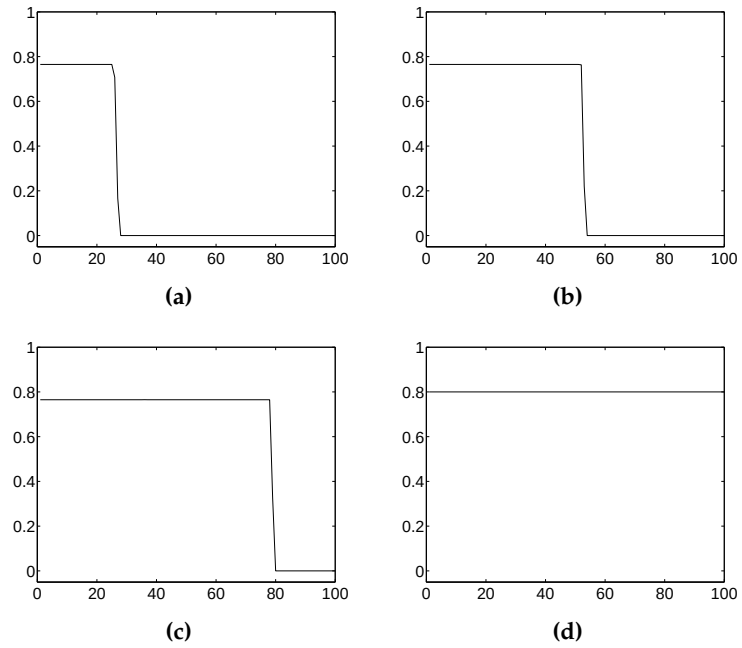

Figure 4.4: Optimal energy profiles with the refined model, (i.e. with convolution). Subfigure a)–d) shows symbol energy versus frame length for 20, 40, 60 and 80 percent of unit energy per symbol, respectively.

error-free, and when the noise increases just beyound the threshold, the error rate increases rapidly. With this in mind, one would expect that the optimal symbol energy distribution for the coded case should be more "thresholded" than the optimal distribution for the uncoded case. More precisely, beyond a certain frame position (that depends on the total amount of energy available for transmitting the frame), the enery used for the remaining symbols should drop off rapidly to zero, and this is exactly what we observe from our model.

As mentioned before, interpolation is used for bounding the error probabilities when the code rate R is larger than the computational cutoff rate R_0, but this interpolation corresponds well to the actual computational behavior of a sequential decoder at high noise levels—the interpolation emulates the actual behavior of FEED observed in simulations, i.e. if the SNR is lowered beneath a certain threshold, the error probability and the time for decoding a frame increases dramatically.

We want to answer the question of why the optimal energy distribution is very nearly a truncation. To simplify the discussion, we consider the basic model (i.e. the model where no convolution is used to "smooth out" the error probabilities). Also, the discussion is not tied specifically to FEED—on the contrary, it will apply to any system that shares the following properties with FEED:

- The system is capable of detecting the first uncorrected error in a frame.
- The system has a *thresholding effect*, i.e. for symbol energies down to a threshold, the probability of an uncorrected symbol error is small, and below the threshold the symbol error probability increases rapidly (towards the final value 0.5, corresponding to zero symbol energy, i.e. corresponding to guessing the symbol value at the decoder).

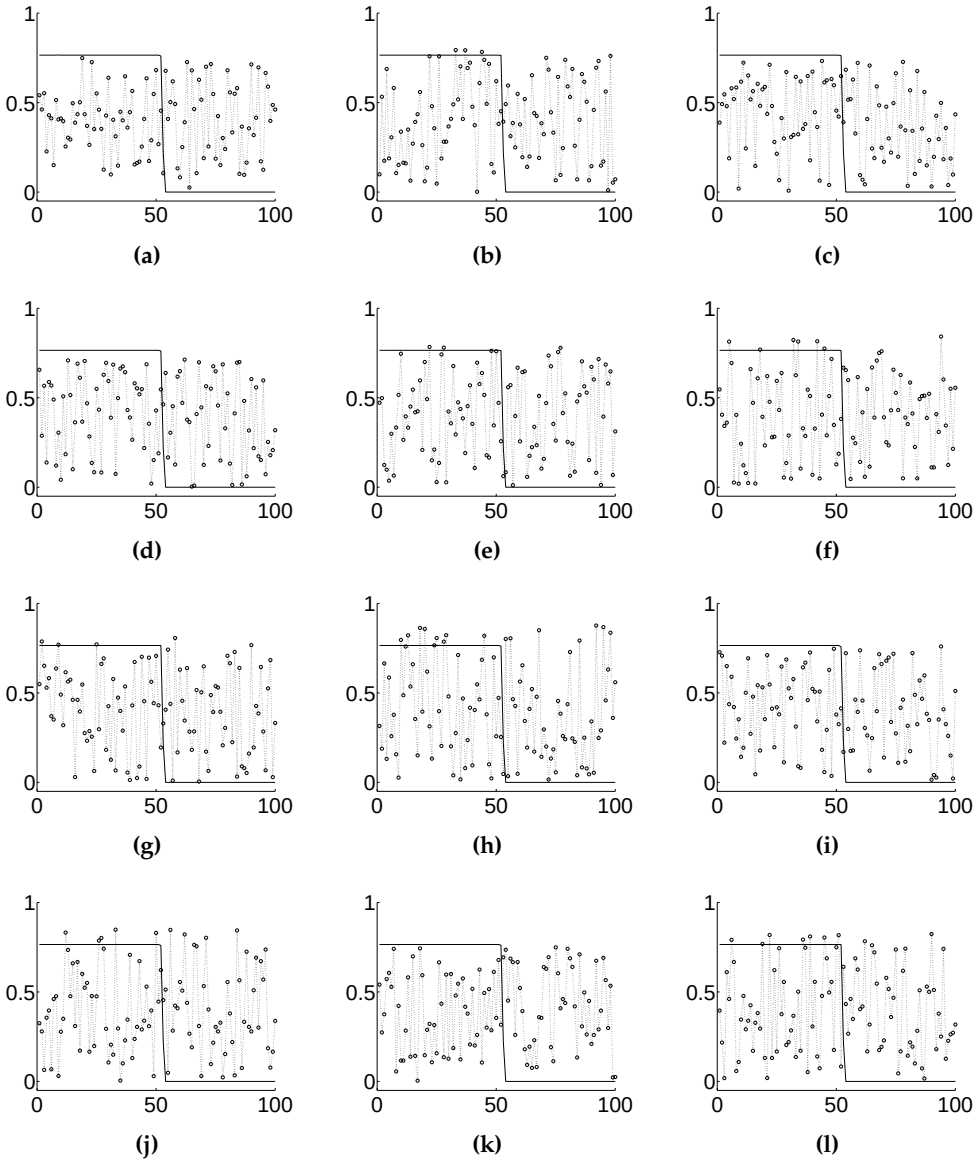

Figure 4.5: Stochastic verification of global optimum; In this example, 12 random energy distributions with average 0.4 (i.e 40%) of unit energy per symbol were used as start values, and in each case, the optimization converged to the same global optimum. The figures show the initial energy distrubutions as dotted curves with circles at the 100 discrete symbols, and the optimum as a continuous curve.

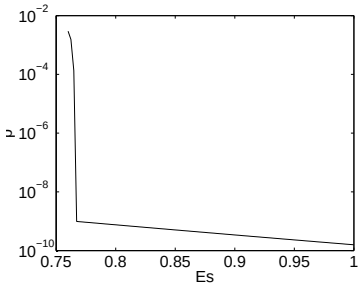

Figure 4.6: Bit error probability versus bit energy

This thresholding effect is an intrinsic property of many systems that utilize error-correcting codes. In an example using sequential decoding of convolutional codes (see Fig. 12.13a) p. 372 of [LJ83]) we see that the bit error probability drops from approx. 10^{-2} at $E_b/N_0 = 4$ dB to approx. 10^{-6} at $E_b/N_0 = 5$ dB. In other words, a difference in symbol energy of approx. 26% (the quotient of the E_b/N_0 values at 5 dB and 4 dB is $10^{1/10} \approx 1.26$) gives a difference in error probability of a factor 10000! Our model behaves in the same way as this example—to see this, consider Fig 4.6.

Recall that the purpose of the optimization was to find the energy distribution that maximizes the error-free length. Our reasoning leads us to a formal statement, that will be tested below.

CONJECTURE: ***Any system that has the two properties listed above will have an optimal energy distribution very close to a truncation***.

The key is to see that *the main factor that governs the expected error-free length for a given energy distribution is the frame index K where the symbol energy drops below a threshold*. The conjecture must of course be tested against our actual optimization results. Again consider Fig 4.6. If our conjecture holds, then the following test should work out correctly: Find the theshold in $P(E_s)$, i.e. the symbol energy E_s where the symbol error probability increases dramatically. For a given total energy in a frame, set the symbol energy value just above the threshold and calculate how many symbols K of the frame can be sent with this energy—this value K should be equal to the value obtained from the optimization, i.e. equal to the frame position where the "truncation" is located. Equivalently, we may just check if the symbol energy obtained from the optimizations is equal to the symbol energy threshold.

Comparing Fig 4.1 and Fig 4.4 to Fig 4.6, we see that this test works out very well—in each case, the symbol energy threshold where the error probability increases dramatically is equal to the symbol energy of the truncated frame obtained from the optimization.

4.4 Conclusions

The main conclusion is that using a truncated symbol energy profile seems (see below) to be optimal in conjunction with FEED. Thus, under heavy noise, the frame should be truncated and all transmitter energy should be spent (equally) on the more important symbols in the beginning of the frame. Since the optimum is truncation instead of a more general symbol energy profile, the effects can be described by a simple example: Assume that the current channel SNR is 1.0 dB. At such a low SNR, the probability of a time-out (i.e. that the convolutional decoder in FEED has to stop before processing the whole code tree corresponding to the received frame) is very high, and FEED will (on average) only be able to recover a few percent of the frame. By instead truncating at 50%, i.e by spending all the energy at the first half of the frame, the effective SNR will increase by 3 dB. With a 3 dB increase in SNR, FEED will (on average) be able to recover everything that is sent, i.e. 50% of the frame. This simple argument is also validated by simulations using FEED, and the reader is referred to [AL02] for a much more extensive discussion/presentation. As a sidenote, the benefits of doubling the energy of the symbols in the truncated frame can, and should, be improved even further by instead using more redundancy. For example, if we send only half as many symbols, we can change the code rate from $\frac{1}{2}$ to $\frac{1}{4}$, i.e. use a more powerful code instead of just transmitting with doubled energy on the (non-truncated) symbols. The optimization merely tells us that at *a fixed code rate*, it is optimal to truncate and "shout" with doubled energy.

As with all mathematical models, the validity of predictions from the model depends on how closely the model resembles reality. Finding the optimal symbol energy distribution for a FEED-capable convolutional encoder is a very complex problem, and the present model ("smoothing" the error probabilities with convolution) is merely one way of attacking the problem. Also, the conclusions are qualitative, not quantitative. This is due to several reasons. For example, the expressions used for the probability of an uncorrected error are bounds, and the bounds are in general not tight. Furthermore, as mentioned before, the problem had to be scaled down to shorten the time required for optimization runs in Matlab, i.e. we simulate a convolutional coder of memory/constraint length ~ 10 bits, and frame lengths ~ 100 bits, instead of the actual FEED situation with memory ~ 100 bits and frame lengths often ~ 1000 bits. Also, the model contains constants that have to be estimated and set in the actual Matlab code. The value of these constants are also scaled when the problem is scaled down as above.

With the above in mind, a few observations which indicate that the model is reasonable are listed:

- The conclusions (which are all qualitative, as stated above) do not rely on a very special choice of the constants in the model. However, choosing too small constants had to be avoided because of numerical stability issues.
- The results for the scaled-down problem are analogous to the results for the actual problem—as mentioned before, the reduction of problem size was done to avoid too lengthy calculations on the computer. However, we have made some full-scale runs and compared with the corresponding scaled-down problem, and in all cases, the conclusions were identical.
- Even though the choice of w, the "convolution window", is rather arbitrary in the model, different window shapes give the same optimum.

- As mentioned before, interpolation is used for bounding the error probabilities when the code rate R is larger than the computational cutoff rate R_0, but this interpolation corresponds well to the actual computational behavior of a sequential decoder at high noise levels—the interpolation emulates the actual behavior of FEED observed in simulations, i.e. if the SNR is lowered beneath a certain threshold, the error probability and the time for decoding a frame increases dramatically.

4.5 A Few Remarks on Discrete Convolution

The purpose of this section is to describe a few practical and notational matters concerning convolution and its use in the refined optimization model.

If f and g are bi-infinite sequences (i.e. $f = [\ldots, f_{-2}, f_{-1}, f_0, f_1, \ldots]$ and similarly for g), their convolution $h = f * g$ is defined by $h_k = (f * g)_k = \sum_n f_n g_{k-n}$, where the sum is over all n that contribute with nonzero terms. The notations $(f * g)_k$ and $(f * g)(k)$ are used interchangeably, depending on the context. Algorithmically, discrete convolution is a straightforward operation. However, different programming languages or different software packages require different indexing etc.—for example, normal C/C++ arrays are indexed from 0, and in Matlab, vectors are indexed from 1.

A couple of remarks on equations (4.12) and (4.13); Note that if the support of w has length m, the convolution in the equations will have length $N + m - 1$ and be defined for values of n larger than N, but we will not use those values since they are not needed in the gradient. The other remark is on implementation—Matlab uses the a slightly different definition of convolution, $w_k = \sum_k u_j v_{k+1-j}$. This is due to the fact that in Matlab, the first component in a vector has index 1, instead of 0.

Chapter 5

Explanations of Some Concepts and Terminology Used in this Paper (in alphabetical order)

ARQ, Automatic Repeat Request Correcting errors by detecting errors at the receiver and requesting retransmission of the erroneous data.

AWGN channel Additive White Gaussian Noise channel.

BSC, Binary Symmetric Channel Special case of a DMC with binary input, binary output and symmetric transition probabilities, i.e. the probability of a 1 being distorted into a 0 is the same as the probability of the opposite. Note that a DMC is by definition memoryless, so the same applies for a BSC.

DHMS, Discrete Hidden Markov Source See Markov chain below.

DMC, Discrete Memoryless Channel A channel with no intersymbol interference and discrete outputs. If the number of outputs exceed 2, then we say that the channel is soft-decision.

FEC, Forward Error Correction Employing error-correcting codes that automatically correct errors detected at the receiver. If there is a communication link from the receiver back to the transmitter, an ARQ scheme may be used instead, see ARQ.

Markov chain A Markov process where the samples take on values from a discrete and countable set Ω_Ψ. Often, Ω_Ψ is the integers or some suitable subset thereof. Markov chains are very useful because they can model a very common phenomenon in technology and science; the finite state machine, e.g. a convolutional encoder is a finite state machine.

(discrete-time) Markov process A random process that satisfies

$$p(\Psi_{k+1} \mid \Psi_k,\ \Psi_{k-1}, \ldots) = p(\Psi_{k+1} \mid \Psi_k), \tag{5.1}$$

(definition from [LM94]). This means that a Markov process is a random process where the sample Ψ_{k+1} is independent of previous samples $\Psi_{k-1}, \ldots$ if the most recent sample Ψ_k is known.

Shannon's noisy channel coding theorem Every channel has a channel capacity C, and for any code rate $\frac{k}{n} = R < C$ $\exists$ codes of rate R that, with maximum likelihood decoding, have an arbitrarily small decoding error probability $P(E)$. In particular, $\exists$ convolutional codes of memory order m such that

$$P(E) \leq 2^{-(m+1)\, n\, E_c(R)} = 2^{-n_A\, E_c(R)}, \tag{5.2}$$

where n_A, as defined in equation (5.2), is called the constraint length. $E_c(R)$ is a positive function of R for $R < C$ and is completely determined by the channel characteristics.

Bibliography

[AL02] Erik Alapää and James P. LeBlanc. FEED encoded data with regressive bit energy. In *Proc. 5^{th} Nordic Signal Processing Symposium, NORSIG 2002*, October 2002.

[BCJR74] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on information theory*, March:284–287, 1974.

[Bin90] J.A.C. Bingham. Multicarrier modulation for data transmission: An idea whose time has come. *IEEE Communication Mag.*, 28:5–14, May 1990.

[Fow00] Martin Fowler. *UML Distilled*. Addison-Wesley, second edition, 2000.

[HSWD00] J. Hagenauer, T. Stockhammer, C. Weiss, and A. Donner. Progressive source coding combined with regressive channel coding on varying channels. In *Proc. 3rd ITG Conference Source and Channel Coding*, pages 123–130, Munich, Germany, January 2000.

[Jos99] Nicolai M. Josuttis. *The C++ Standard Library*. Addison-Wesley, 1999.

[JZ99] Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of Convolutional Coding*. IEEE Press, 1999.

[LJ83] Shu Lin and Daniel J. Costello Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, N.J., 1983.

[LM94] Edward A. Lee and David G. Messerschmitt. *Digital Communication*. KAP, Kluwer Academic Publishers, second edition, 1994.

[Mas72] James L. Massey. Variable-length codes and the Fano metric. *IEEE Transactions on information theory*, IT-18:196–198, 1972.

[SL02] Martin Sehlstedt and James P. LeBlanc. Bit energy distribution effects in progressively encoded video frame transmissions. In *Proc. IEEE Int'l Conf. on Multimedia and Exhib.*, Aug. 28 2002.

[Vit95] A.J. Viterbi. *CDMA: Principles of Spread Spectrum Communication*. Addison-Wesley, 1995.

[WC95] Fu-Quan Wang and Daniel J. Costello. Probabilistic construction of large constraint length trellis codes for sequential decoding. *IEEE Transactions on Communications*, 43:2441, September 1995.

[WSH01] C. Weiss, T. Stockhammer, and J. Hagenauer. The far end error decoder with application to image transmission. In *Proc. IEEE Globecom 2001*, pages 1405–1409, San Antonio, TX, November 2001.

Index

FEED ENCODED DATA WITH REGRESSIVE BIT ENERGY

Erik Alapää

Dept. of Mathematics,
Chalmers University of Technology,
Gothenburg, Sweden
alapaa@math.chalmers.se

James P. LeBlanc

Div. of Signal Processing,
Luleå University of Technology,
Luleå, Sweden
leblanc@sm.luth.se

ABSTRACT

The efficient use of FEED, a new type of sequential decoder well suited for transmission of progressively encoded multimedia sources, is investigated. The pricincipal feature of the FEED decoder is that it is capable of detecting the position of the first uncorrected channel error, thus improving e.g. image quality by enabling the source decoder to use only the error-free part of the transmitted frame. The simulations in this article show that in the case of heavy noise, the performance of FEED can be substantially improved by a simple truncation scheme—by concentrating the available transmitter power to a truncated subframe, the average length of the error-free part of the received frame increases radically.

1. INTRODUCTION

Modern source compression schemes for multimedia (sound, still images, video etc.) may be *progressive*. To explain this term, consider the compression of an audio signal using some suitable transform such as the FFT or a wavelet transform. An audio frame is converted into a set of coefficients, which are sorted in decreasing size order. The ones that are below a given threshold are truncated, thus acheiving the compression. Consider the transmission of the remaining coefficients (in decreasing size order) over a noisy communication channel and reconstruction of the audio signal at the receiver end, with the following observations:

- One can reconstruct a reasonable approximation of the signal if the most important (i.e. largest) coefficients in the frame have been received correctly.
- Moving further into the frame, the reconstructed signal quality increases with the number of correctly received coefficients.
- Due to variable run-length encoding, if one coefficient has been corrupted by noise, the following coefficients in the frame can actually degrade the reconstructed quality [2].

The complete communications system should be designed to accomodate multimedia signals compressed with progressive source coding schemes. One way of doing this can briefly be described as follows:

1. Use *FEED* [6], a new sequential decoder for convolutional codes that can locate the frame position of the first uncorrected error.
2. Combine FEED with regressive redundancy, i.e. protect the bits (representing the coefficients) with regressive redundancy according to their importance.

This regressive redundancy is usually achieved by puncturing. This paper presents a modification of the approach in [6]. Herein the regressive redundancy part acheived by puncturing is replaced with a transmitter that spends more transmission energy on more important bits in a frame.

Partially supported by Ericsson Erisoft, the Swedish IT Institute Internet3 program and EU Mål 1. Also, special thanks to Drs. Stockhammer and Weiss from Prof. Hagenauer's group at Munich Univ. of Tech. for their support on FEED questions, sharing prepub. papers etc.

2. DESCRIPTION OF THE FEED CONCEPT

Shannon's noisy channel coding theorem states that when the transmission rate is below the channel capacity, arbitrarily small error probabilities can be obtained by using an encoder with large enough memory. However, to decode long-memory convolutional codes one cannot use the Viterbi maximum likelihood algorithm, since its complexity grows exponentially with the constraint length (which is proportional to the encoder memory). An alternative is then to use sequential decoding [3], since sequential decoding complexity is essentially independent of encoder memory. However, in practice sequential decoders may sometimes take a very long time to decode noisy frames, which can result in a time-out before the whole frame is decoded. The standard solution to this has been to declare an erasure and possibly request a resend. The idea with FEED is to avoid these erasures and try to salvage the useful portion of the partially decoded frame.

To understand how this is done, recalling a few facts about sequential decoders is necessary. When analyzing sequential decoders of convolutional codes, it is useful to view the code as a *code tree*. For example, a rate 1/2 convolutional code and a frame length of 1152 information bits corresponds to a full binary code tree of height 1152 with 2^{1152} leaf nodes (in practice, the number of leaf nodes will be somewhat lower, since the "tail" of any transmitted frame is a sequence of zeros that forces the convolutional encoder back to its zero state). The sequential decoder starts at the root node and compares different paths through the code tree by looking at the received sequence and computing path metrics in the tree, usually using the Fano metric [3], [4]. If the decoding process is stopped before the decoder is finished with a frame, one has a *partially explored code tree*, as depicted in Fig. 1. In this example, the current path estimate could be the path leading from the root to node *B*, and the intent is to determine the reliability of this path, and all its subpaths beginning at the root.

The heart of the FEED concept is this path reliability calculation, which has roots in [4] and [1]. The algorithm is described by the inventors of FEED in [6]. In Appendix A a full derivation of the path reliability calculation is given. The appendix is provided for the convenience of interested readers and essentially

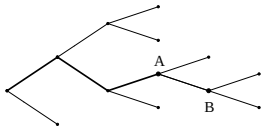

Figure 1: Partially explored code tree

consists of a combination of the relevant parts of [4] and [6], with a coherent notation etc. For the reader who wants a brief, intuitive explanation of the path reliability calculation, an overview follows here.

In considering Fig. 1, let $R(A)$ denote the path reliability for the A-path, i.e. the subpath going from the root to node A. $R(A)$ is calculated as

$$R(A) = \text{weight of } A\text{-path} \cdot \frac{\sum \text{weight of all subpaths stemming from } A}{\sum \text{weight of all paths in part. expl. code tree}}, \quad (1)$$

where "weight" simply means the Fano path metric. That this is a good reliability measure can be understood intuitively from how a Fano decoder operates. If the paths that have the A-path in common constitute a considerable portion of the partially explored code tree, this means that the decoder has spent a lot of time in the subtree stemming from node A, indicating that the A-path is reliable. Performing this reliability calculation for all subpaths that lead to node B and noting when a significant reliability drop occurs, one obtains an estimate of where the first uncorrected error occured. In our simulations, this estimate has been verified to be very accurate, allowing to salvage correctly decoded parts of timed-out frames.

3. OUTLINE OF THE METHOD

The idea of this paper is to combine FEED with the use of regressive bit energy instead of regressive redundancy. Regressive redundancy is often accomplished by puncturing, which means that finding the optimal regressive redundancy may be viewed as optimization over a discrete space. Furthermore if, as is common, puncturing is done in hardware, this might further limit the flexibility which is desirable as channel SNR fluctuates over a broad range. In contrast, a regressive bit energy profile may allow a more malleable solution. See also [5]. Also note that the two methods (puncturing and adjustable bit energy) may be used in concert.

4. RESULTS

For the simulations, a FEED implementation that was developed by the first author for Ericsson Erisoft is used. A systematic ODP (Optimum Distance Profile) convolutional code of rate $R = 1/2$ with 96 bit memory (in effect, 95 bits plus "current bit", in order to align with 32-bit word boundaries on the target CPUs), a frame length of 1152 information bits and BPSK on an AWGN discrete, memoryless channel (DMC) are used. The DMC is binary-input, 256-output (implementation note: the 256 outputs correspond to an `unsigned char` in C/C++ for most CPU architectures). The sequential decoder uses the Fano algorithm for the actual decoding part, with some added bookkeeping to be able to do the FEED path reliability calculation as a post-processing step. The system is depicted in Fig. 2.

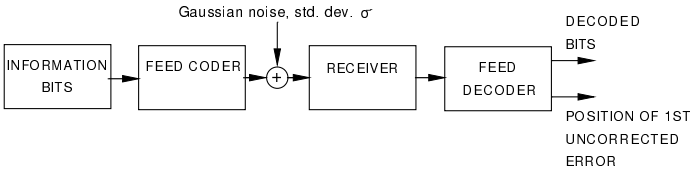

Figure 2: System block diagram

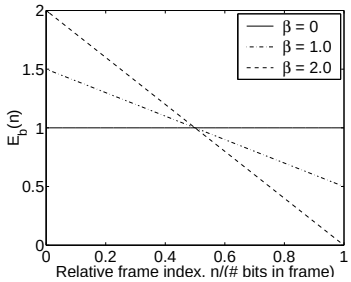

Figure 3: Regressive $E_b(n)$ profiles for three different slopes (slope $= -\beta$)

4.1. Initial attempt—linearly regressive bit energy

For simplicity, a linearly regressive energy per bit $E_b(n)$ is used. Let n_{syms} denote the number of (code) bits in a frame, and let n denote the bit index. E_b can then be written $E_b(n) = -\beta n/n_{syms} + c$, where the parameter β governs the slope of the E_b lines and c is a constant chosen so that the total bit energy in a frame is independent of β, see Fig. 3. Note that we do not consider a linearly regressive profile to be optimal, it is just a simple model for experimentation. See also [5] where different $E_b(n)$ profiles are investigated.

Fig. 4 shows average recovered frame length versus β for five different SNRs (recall that small β values correspond to almost flat $E_b(n)$ profiles, and larger β correspond to more regressive $E_b(n)$). In this figure, and in all other figures in this paper, the SNR values are in dB. Each data point is an average of 30 independent runs of the decoder with fixed average SNR and β, and each run is a simulation of 20 frames. As may be expected, for low SNRs, it is best to use a higly regressive power profile placing most of the energy in the early part of the frame, while for high SNRs, a flat power profile gives the longest recovered frame lengths. Since the transition from regressive to flat optimum occurs for an SNR between 1.5 and 2.0 dB, a more detailed plot of this transition is included in Fig. 5.

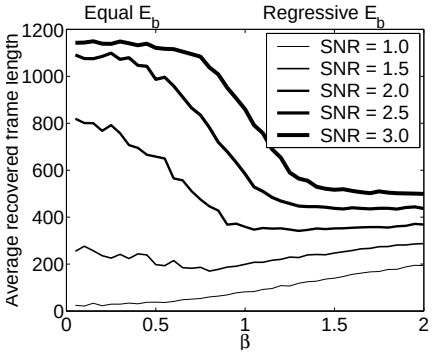

Figure 4: Average recovered frame length versus β

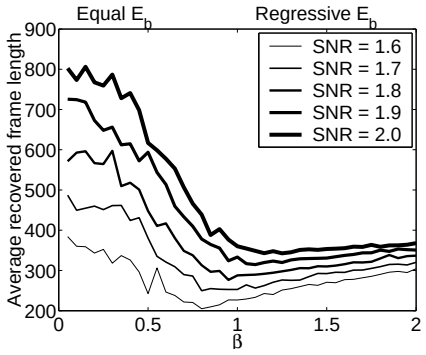

Figure 5: Average recovered frame length versus β

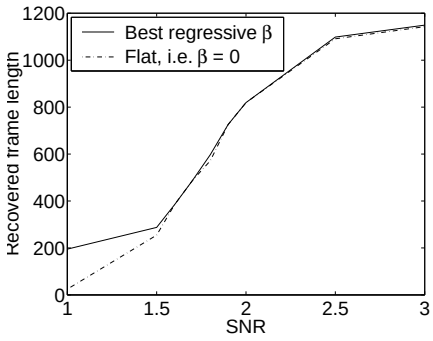

Figure 6: Average recovered frame length versus SNR for best (linearly) regressive and flat $E_b(n)$

A complementary view of the data is shown in Fig. 6. For each SNR, the β that gave the longest recovered frame length was chosen for the regressive curve in the figure. The two curves demonstate that the regressive E_b profiles give much larger recovered frame lengths for low SNRs (between 1 and 1.5 dB), but for higher SNRs, there is no benefit from using a linearly regressive bit energy.

4.2. Truncation instead of linearly regressive bit energy

For low SNRs, it is reasonable to assume that a better approach than a linearly regressive bit energy would be to use truncation, i.e. concentrate all the available transmitter power to a subframe to enable longer recovered frame lengths. This assumption is further supported by a theoretical investigation in the case of no channel coding—in [5], it is shown that for progressively encoded sources and low SNRs, the optimal bit energy distribution goes to zero before the end of the frame. It is also shown that using a hard truncation instead of the optimal bit enery distribution causes essentially no performance loss (the difference to the optimal case is only a few percent).

The theoretical discussion above is affirmed by simulations—Fig. 7 demonstrates that even at an SNR of 1 dB, almost 800 of 1152 bits can (on average) be recovered by truncating at 70%, i.e. by concentrating all transmitter power to the first 70% of the frame. Also worthy of mentioning is the fact that truncation not only increases the average, it also decreases the "spread"—the recovered frame length of the worst-case frames, i.e. the timed-out frames, increases radically.

The performance could be improved upon further by refining the transmission system, e.g:

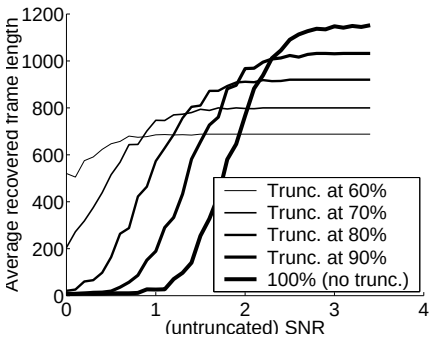

Figure 7: Average recovered frame length versus SNR for different frame truncation percentages

- It is well known that sequential decoder performance is sensitive to metric table incorrectness, i.e. optimal decoder performance is contingent on the use of a metric table that corresponds to current channel SNR. For simplicity, and to avoid distorting the comparison with linearly regressive bit energy, the metric table of the decoder whas not modified when concentrating all transmitter power to a subframe, even though this power concentration increases the effective SNR for the transmitted symbols.
- The truncation of course implies that fewer symbols are actually transmitted over the channel. This implies a lower bitrate, so it could be feasible to decrease the used bandwidth, thereby decreasing the channel noise power and getting an additional SNR increase.

5. CONCLUSION

The simulations demonstrate that for the lowest SNRs, it is advantageous to truncate, i.e. to spend all of the transmitter's energy budget in the beginning of the frame, since such a truncated $E_b(n)$ profile can significantly increase the recovered frame length for noisy frames. For example, if a transmission system detects that it is in a low SNR situation, it could truncate early in the frame, thereby increasing the correctly received frame length. When the system detects increasing SNR, the truncation would be adjusted to occur later in the frame, allowing the decoder to recover larger parts of the frame.

A scheme such as the one above requires that the receiver somehow detects the current SNR and reports it back to the transmitter, i.e. a low bit-rate return link from receiver to transmitter would be necessary. Also, the question is how FEED should obtain the current SNR. One way is provided by the curve for the flat $E_b(n)$ profile in Fig. 6—the inverse of the function corresponding to the curve is a function that maps recovered frame lengths to current SNRs, and such a function could for instance be stored as a lookup table in the decoder.

As a sidenote, it should be pointed out that the FEED decoder also is inherently adaptive to varying noise—for fixed $E_b(n)$ profile, the average recovered frame length for timed-out frames increases radically when the SNR increases. This of course is due to the fact that with increased SNR, the sequential decoder needs fewer decoder cycles to work its way through the code tree. Therefore, in a given time, the decoder manages to move further into the tree in case of high SNR, so if a time-out occurs, the reliability calculation then allows FEED to recover a larger part of

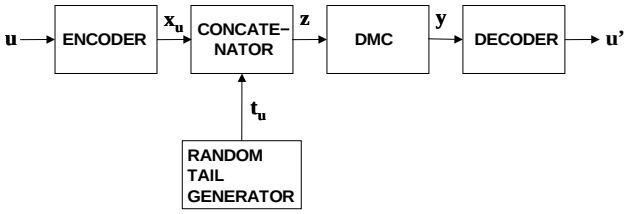

Figure 8: Transmission system for variable-length code

the frame, see also [2].

A. DERIVATION OF THE PATH RELIABILITY FORMULA

A.1. A Connection Between Variable-Length Codes and the Fano Metric

Consider the transmission system shown in Fig. 8 for a variable-length code. Let $\mathcal{U}$ be a set of M messages (information words). To each information word $\mathbf{u} \in \mathcal{U}$ corresponds a code word $\mathbf{x_u}$

$$\mathbf{x_u} = [x_{\mathbf{u},1}, x_{\mathbf{u},2}, \ldots, x_{\mathbf{u},n_\mathbf{u}}]$$

of length $n_\mathbf{u}$. The code symbols $x_{\mathbf{u},j}$, $j = 1, \ldots, n_\mathbf{u}$ belong to some alphabet $\mathcal{X}$, e.g. if we have a rate 1/2 binary convolutional code each bit (in this case, with one input to the encoder, an information word symbol is one bit) in an information word $\mathbf{u}$ would correspond to a two-bit code word symbol $x_{\mathbf{u},j} \in \mathcal{X}$. Now, let $\mathcal{S}$ denote the set

$$\mathcal{S} = \{\mathbf{x_u} \mid \mathbf{u} \in \mathcal{U}\}.$$

Then, $\mathcal{S}$ is our variable-length code, and we can send code words from $\mathcal{S}$ through a DMC. The goal of the variable length decoder is to estimate the transmitted variable-length codeword. The estimate is based on the received output sequence $\mathbf{y} = \mathbf{y}_{(0:T]}$, and the goal is to minimize the (*a posteriori*) error probability. Therefore, (since $\mathbf{y}$ is fixed, i.e. it is an observation that can be viewed as a fixed quantity for the analysis here), the estimate $\hat{\mathbf{u}}$ is taken as the message word $\mathbf{u} \in \mathcal{U}$ that maximizes $\Pr\{\mathbf{u}, \mathbf{y}\}$.

To calculate the maximizing estimate, we use the theory described in [4]. Consider the abstract transmission system depicted in Fig. 8. Denote the maximal codeword length by T. The message $\mathbf{u}$ of probability $P_\mathbf{u}$ selects the codeword

$$\mathbf{x_u} = [x_{\mathbf{u},1}, x_{\mathbf{u},2}, \ldots, x_{\mathbf{u},n_\mathbf{u}}],$$

to which is added a "random tail"

$$\mathbf{t_u} = [t_{\mathbf{u},1}, t_{\mathbf{u},2}, \ldots, t_{\mathbf{u},T-n_\mathbf{u}}],$$

and the message plus the random tail together form a fixed-length input sequence

$$\mathbf{z} = [z_1, z_2, \ldots, z_T] = [\mathbf{x_u}, \mathbf{t_u}].$$

This sequence $\mathbf{z}$ is then transmitted over the DMC. The random tail $\mathbf{t_u}$ is assumed to be selected statistically independently of $\mathbf{x_u}$, and we also assume that the digits in $\mathbf{t_u}$ are chosen independently according to a probability measure $Q()$ over the channel input alphabet, i.e,

$$\Pr\{\mathbf{t_u} \mid \mathbf{x_u}\} = \Pr\{\mathbf{t_u}\} = \prod_{k=1}^{T-n_\mathbf{u}} Q(t_k).$$

The introduction of the random tail $\mathbf{t_u}$ might confuse the reader at first, but its use will become clearer below. It suffices to think of $\mathbf{t_u}$ as either a convenient device for normalizing the number of received digits that must be considered in the decoding process, or as the digits resulting from subsequent encodings of further messages in a randomly selected code.

Let $\mathbf{y} = [y_1, y_2, \ldots, y_T]$ denote the received word. By the definition of a DMC we have

$$\Pr\{\mathbf{y} \mid \mathbf{z}\} = \prod_{t=1}^{n_\mathbf{u}} p_c(y_i \mid x_{\mathbf{u},i}) \prod_{j=1}^{T-n_\mathbf{u}} p_c(y_{n_\mathbf{u}+j} \mid t_j),$$

where $p_c(\cdot \mid \cdot)$ denotes the transition structure of the channel.

The joint probability of sending the message $\mathbf{u}$, adding the random tail $\mathbf{t_u}$ and receiving $\mathbf{y}$ can then be written

$$\begin{aligned}\Pr\{\mathbf{u}, \mathbf{t_u}, \mathbf{y}\} &= P_\mathbf{u} \Pr\{\mathbf{t_u} \mid \mathbf{x_u}\} \Pr\{\mathbf{y} \mid [\mathbf{x_u} \mathbf{t_u}]\} \\ &= P_\mathbf{u} \prod_{i=1}^{n_\mathbf{u}} p_c(y_i \mid x_{\mathbf{u},i}) \prod_{k=1}^{T-n_\mathbf{u}} Q(t_k) \prod_{j=1}^{T-n_\mathbf{u}} p_c(y_{n_\mathbf{u}+j} \mid t_j).\end{aligned}$$

If we replace the "dummy" index k on the last line of the previous equation by j, we get

$$\Pr\{\mathbf{u}, \mathbf{t_u}, \mathbf{y}\} = P_\mathbf{u} \prod_{i=1}^{n_\mathbf{u}} p_c(y_i \mid x_{\mathbf{u},i}) \prod_{j=1}^{T-n_\mathbf{u}} p_c(y_{n_\mathbf{u}+j} \mid t_j) Q(t_j).$$

Summing over all possible random tails and setting

$$P_0(y_i) = \sum_{t_k} p_c(y_i \mid t_k) Q(t_k), \tag{2}$$

gives

$$\begin{aligned}\Pr\{\mathbf{u}, \mathbf{y}\} &= P_\mathbf{u} \prod_{i=1}^{n_\mathbf{u}} p_c(y_i \mid x_{\mathbf{u},i}) \prod_{j=1}^{T-n_\mathbf{u}} \left(\sum_{t_k} p_c(y_{n_\mathbf{u}+j} \mid t_k) Q(t_k) \right) \\ &= P_\mathbf{u} \prod_{i=1}^{n_\mathbf{u}} p_c(y_i \mid x_{\mathbf{u},i}) \prod_{j=1}^{T-n_\mathbf{u}} P_0(y_{n_\mathbf{u}+j}). \qquad (3)\end{aligned}$$

Thus, $P_0()$ is the probability measure on the channel output alphabet when the probability distribution on the channel input is $Q()$ as above. Now, given $\mathbf{y}$, the optimal decoding rule (optimal in the sense that the rule minimizes the probability of an erroneous decision) is to choose the message $\hat{\mathbf{u}}$ that maximizes $\Pr\{\mathbf{u}, \mathbf{y}\}$, which is equivalent to maximizing

$$\frac{\Pr\{\mathbf{u}, \mathbf{y}\}}{\prod_{i=1}^{T} P_0(y_i)},$$

since the denominator does not depend on the message $\mathbf{u}$. Taking logarithms, and using (2) and (3), we obtain the log-likelihood ratio

$$\begin{aligned}L(\mathbf{u}, \mathbf{y}) &= \log \left[\Pr\{\mathbf{u}, \mathbf{y}\} / \prod_{i=1}^{T} P_0(y_i) \right] \\ &= \log \left[P_\mathbf{u} \prod_{i=1}^{n_\mathbf{u}} \frac{p_c(y_i \mid x_{\mathbf{u},i})}{P_0(y_i)} \right] \\ &= \log(P_\mathbf{u}) + \sum_{i=1}^{n_\mathbf{u}} \left[\log \frac{p_c(y_i \mid x_{\mathbf{u},i})}{P_0(y_i)} \right] \\ &= \sum_{i=1}^{n_\mathbf{u}} \left[\log \frac{p_c(y_i \mid x_{\mathbf{u},i})}{P_0(y_i)} + \frac{1}{n_\mathbf{u}} \log P_\mathbf{u} \right].\end{aligned}$$

The probability $\Pr\{\mathbf{u},\mathbf{y}\}$ can equally well be written as

$$\begin{aligned}\Pr\{\mathbf{u},\mathbf{y}\} &= C(\mathbf{y})\cdot\exp(\Lambda_{\mathbf{u}}) \\ &= C(\mathbf{y})\cdot\exp(\sum_{j=1}^{n_{\mathbf{u}}}\lambda_{\mathbf{u},j}),\end{aligned} \tag{4}$$

where C is a constant that only depends on the received word $\mathbf{y}$, and the metric increment $\lambda_{\mathbf{u},j}$ for each received symbol is

$$\lambda_{\mathbf{u},j} = \log\frac{p_c(y_j \mid x_{\mathbf{u},j})}{\Pr(y_j)} + \frac{1}{n_{\mathbf{u}}}\log\Pr\{\mathbf{u}\}, \tag{5}$$

where $\Pr\{\mathbf{u}\}(=P_{\mathbf{u}})$ is the (*a-priori*) probability of the message $\mathbf{u}$ (the constant C is of course just $\prod_{i=1}^{T}P_0(y_i)$). As will be shown in section A.3, in the common case of equiprobable information words, equation (5) defines the well-known *Fano metric*. This metric is used for sequential decoding of convolutional codes, e.g. in the Fano algorithm. Taking logarithms does not alter the results of maximization since the logarithm is a strictly increasing function. As an additional benefit, the fact that we take logarithms gives us better numerical stability—in the original BCJR algorithm [1] and in the algorithm described in the earlier sections, we wind up with multi-factor products of probabilities, and those products quickly become small, which can cause numerical instability. Logarithms convert these products to sums, thereby avoiding having to deal with very small numbers.

A.2. Path Reliability for Variable-Length Codes

Given the entire received sequence $\mathbf{y}$ and a subpath $\mathbf{x}_{(0:t]}$ representing an estimate of a transmitted codeword, we want to calculate the reliability of the subpath. We begin by defining a subset $\mathcal{U}(\mathbf{x}_{(0:t]})$ of the information words $\mathcal{U}$ as $\mathcal{U}(\mathbf{x}_{(0:t]}) = \{\mathbf{u}' \in \mathcal{U} \mid \forall_{0<\tau\le t}\ x_{\mathbf{u}',\tau} = x_\tau\}$. We then define a corresponding subset $\mathcal{S}(\mathbf{x}_{(0:t]})$ of the codewords $\mathcal{S}$ as $\mathcal{S}(\mathbf{x}_{(0:t]}) = \{\mathbf{x}_{\mathbf{u}'} \in \mathcal{S} \mid \mathbf{u}' \in \mathcal{U}(\mathbf{x}_{(0:t]})\}$. The reliability of the subpath $\mathbf{x}_{(0:t]}$ can now be written as

$$\begin{aligned}R_{\mathbf{y},\mathcal{U}}(\mathbf{x}_{(0:t]}) &= \frac{\sum_{\mathbf{x}'\in\mathcal{S}(\mathbf{x}_{(0:t]})}\Pr(\mathbf{x}',\mathbf{y})}{\sum_{\mathbf{x}'\in\mathcal{S}}\Pr(\mathbf{x}',\mathbf{y})} \\ &= \frac{\sum_{\mathbf{u}'\in\mathcal{U}(\mathbf{x}_{(0:t]})}\Pr(\mathbf{u}',\mathbf{y})}{\sum_{\mathbf{u}'\in\mathcal{U}}\Pr(\mathbf{u}',\mathbf{y})}.\end{aligned}$$

Using (4) and simplifying we get

$$\begin{aligned}&R_{\mathbf{y},\mathcal{U}}(\mathbf{x}_{(0:t]}) \\ &= \frac{\sum_{\mathbf{u}'\in\mathcal{U}(\mathbf{x}_{(0:t]})}\Pr(\mathbf{u}',\mathbf{y})}{\sum_{\mathbf{u}'\in\mathcal{U}}\Pr(\mathbf{u}',\mathbf{y})} \\ &= \text{[insert (4)]} \\ &= \frac{\sum_{\mathbf{u}'\in\mathcal{U}(\mathbf{x}_{(0:t]})}C(\mathbf{y})\cdot\exp(\sum_{j=1}^{n_{\mathbf{u}'}}\lambda_{\mathbf{u}',j})}{\sum_{\mathbf{u}'\in\mathcal{U}}C(\mathbf{y})\cdot\exp(\sum_{j=1}^{n_{\mathbf{u}'}}\lambda_{\mathbf{u}',j})} \\ &= \text{[use the definition of } \mathcal{U}(\mathbf{x}_{(0:t]}) \text{ i.e. common subpath]} \\ &= \underbrace{\exp(\sum_{j=1}^{t}\lambda_{\mathbf{u},j})}_{\text{common subpath}}\cdot\frac{\sum_{\mathbf{u}'\in\mathcal{U}(\mathbf{x}_{(0:t]})}\exp(\sum_{j=t+1}^{n_{\mathbf{u}'}}\lambda_{\mathbf{u}',j})}{\sum_{\mathbf{u}'\in\mathcal{U}}\exp(\sum_{j=1}^{n_{\mathbf{u}'}}\lambda_{\mathbf{u}',j})}.\end{aligned} \tag{6}$$

Equation (6) can be used to calculate the reliability of any decoded path $\hat{\mathbf{x}}_{(0:\hat{\mathbf{u}}]}$. If we let t vary between 1 and $n_{\hat{\mathbf{u}}}$, we get a vector of reliabilities of subpaths of the decoded path:

$$\begin{aligned}&\mathbf{R}_{\mathbf{y},\mathcal{U}}(\hat{\mathbf{u}}) \\ &= \mathbf{R}_{\mathbf{y},\mathcal{U}}(\hat{\mathbf{x}}) \\ &= \{R_{\mathbf{y},\mathcal{U}}(\hat{\mathbf{x}}_{(0:1]}), R_{\mathbf{y},\mathcal{U}}(\hat{\mathbf{x}}_{(0:2]}), \ldots, R_{\mathbf{y},\mathcal{U}}(\hat{\mathbf{x}}_{(0:n_{\hat{\mathbf{u}}}]})\}.\end{aligned}$$

A.3. Using the Theory for Variable-Length Codes to Calculate the Path Reliability of Sequential Decoding

If a sequential decoder, for example a Fano decoder, has to stop before it is finished (e.g. due to timing constraints), usually the partially decoded frame has to be discarded. Instead, we will make use of the partially decoded data using the theory developed in [6].

The connection tho the theory for variable-length codes is essentially this important observation: When the decoder has to stop before it is finished, *we have a partially explored code tree that can be viewed as a fully explored code tree for a code with variable-length codewords*! A codeword corresponding to an information word $\mathbf{u}$ is denoted by by $\mathbf{x}_{\mathbf{u}}$, and we have

$$\mathbf{x}_{\mathbf{u}} = \mathbf{x}_{\mathbf{u}(0:n_{\mathbf{u}}]} = [(x_{\mathbf{u},1}^{(1)},\ldots,x_{\mathbf{u},1}^{(n)}),\ldots,(x_{\mathbf{u},n_{\mathbf{u}}}^{(1)},\ldots,x_{\mathbf{u},n_{\mathbf{u}}}^{(n)})],$$

where n denotes the number of codeword bits per information word symbol, e.g. $(x_{\mathbf{u},4}^{(1)},\ldots,x_{\mathbf{u},4}^{(n)})$ is the n bits in $\mathbf{x}_{\mathbf{u}}$ corresponding to the fourth symbol u_4 in the information word $\mathbf{u} = \mathbf{u}_{(0:n_{\mathbf{u}}]} = [u_1,\ldots,u_4,\ldots,u_{n_{\mathbf{u}}}]$. We denote the set of information words corresponding to the partially explored code tree by $\mathcal{U}$. The joint probability of the received sequence $\mathbf{y}$ and a path $\mathbf{x}_{\mathbf{u}}$ can be written as an expression of the same form as (4):

$$\begin{aligned}\Pr\{\mathbf{x}_{\mathbf{u}},\mathbf{y}\} &= \Pr\{\mathbf{u},\mathbf{y}\} = C(\mathbf{y})\cdot\exp(\Lambda_{\mathbf{u}}) \\ &= C(\mathbf{y})\cdot\exp\big[\sum_{j=1}^{n_{\mathbf{u}}}\lambda_t\big(s(\mathbf{u}_{(0:t-1]}),s(\mathbf{u}_{(0:t]})\big)\big],\end{aligned}$$

where $s(\mathbf{u}_{(0:t]})$ is the state of the encoder after encoding the information sequence $\mathbf{u}_{(0:t]}$. Observe that the notation

$$\lambda_j\big(s(\mathbf{u}_{(0:j-1]}),s(\mathbf{u}_{(0:j]})\big)$$

of course means the same as $\lambda_{\mathbf{u},j}$.

In the case where one information word symbol is one bit, assuming that the information bits are independent and equally likely to be zeros or ones, the *a priori* probability that the decoder followed the path $\mathbf{u}$ is

$$\Pr\{\mathbf{u}\} = 2^{-n_{\mathbf{u}}}. \tag{7}$$

Using (7), (5) becomes

$$\lambda_{\mathbf{u},j} = \log\frac{p_c(y_j \mid x_{\mathbf{u},j})}{\Pr(y_j)} - 1.$$

In the derivation above, there was one codeword symbol for each information word symbol. If we instead let $n_{\mathbf{u}}$ denote the number of *bits* in the codeword and do a "bitwise" derivation, where the ratio of the number of info bits to the number of code bits equals the code rate R, we would get

$$\Pr\{\mathbf{u}\} = 2^{-Rn_{\mathbf{u}}},$$

and the kth step "bit metric" would be

$$\lambda_{\mathbf{u},k} = \log\frac{r(y_k \mid x_{\mathbf{u},k})}{\Pr(y_k)} - R.$$

with $r(\cdot \mid \cdot)$ the "bit transition probability" for the channel. This "bit metric" is the well-known *Fano metric* for sequential decoding.

It is important to note that the theory developed so far is not restricted to a DMC—it could equally well have been developed for a fading channel. In that case, the metric increment becomes

$$\begin{aligned}\lambda_t(m,m') &= n\log(2) - \sum_{i=1}^{n} \log\left(1+\exp(-\psi_t^{(i)} x_t^{(i)})\right) \\ &\quad - \log(1+\exp(-L(u_t)\cdot u_t)),\end{aligned}$$

where, if $a_t^{(i)}$ denotes the fading amplitude and $\frac{E_s}{N_0}$ the SNR, $\psi_t^{(i)} = 4a_t^{(i)}\frac{E_s}{N_0}y_t^{(j)}$. The interested reader is referred to [6] for details.

To sum up, we have finally arrived at the expression for calculating the path reliability in our convolutional decoder:

$$\begin{aligned}&R_{\mathbf{y},\mathcal{U}}(\mathbf{u}_{(0:t]}) \\ &= \overbrace{\exp\Big(\sum_{j=1}^{t}\lambda_j(s(\mathbf{u}_{(0:j-1]}),s(\mathbf{u}_{(0:j]}))\Big)}^{\text{common subpath}} \\ &\quad\cdot\frac{\sum_{\mathbf{u}'\in\mathcal{U}(\mathbf{u}_{(0:t]})}\exp\left[\sum_{j=t+1}^{n_{\mathbf{u}'}}\lambda_j\left(s(\mathbf{u}'_{(0:j-1]}),s(\mathbf{u}'_{(0:j]})\right)\right]}{\sum_{\mathbf{u}'\in\mathcal{U}}\exp\left[\sum_{j=1}^{n_{\mathbf{u}'}}\lambda_j\left(s(\mathbf{u}'_{(0:j-1]}),s(\mathbf{u}'_{(0:j]})\right)\right]},\end{aligned} \quad (8)$$

compare (8) with (1). This brings us back to our goal—a FEED-capable decoder. Suppose that the decoder receives a time-out before finishing, i.e. before reaching a leaf node of the code tree. We then have a partially explored code tree and an estimate $\hat{\mathbf{u}}$ of (part of) the transmitted sequence. The decoder then computes the subpath reliabilities of all subpaths of $\hat{\mathbf{u}}$ (that begin at the root node). When a large drop in the path reliability occurs, this is an indication that an uncorrected error exists, and FEED delivers the path estimate, up to this first uncorrected error, as its partially decoded frame. Here, we can also see the reason for the name FEED, "Far End Error Decoder"—the decoder attempts to decode as long towards the far end of the frame as possible, and then (given timing constraints etc.) delivers only the error-free part of the decoded frame.

REFERENCES

[1] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on information theory*, March:284–287, 1974.

[2] J. Hagenauer, T. Stockhammer, C. Weiss, and A. Donner. Progressive source coding combined with regressive channel coding on varying channels. In *Proc. 3rd ITG Conference Source and Channel Coding*, pages 123–130, Munich, Germany, January 2000.

[3] Shu Lin and Daniel J. Costello Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, N.J., 1983.

[4] James L. Massey. Variable-length codes and the Fano metric. *IEEE Transactions on information theory*, IT-18:196–198, 1972.

[5] Martin Sehlstedt and James P. LeBlanc. Bit energy distribution effects in progressively encoded video frame transmissions. In *Proc. IEEE Int'l Conf. on Multimedia and Exhib.*, Aug. 28 2002.

[6] C. Weiss, T. Stockhammer, and J. Hagenauer. The far end error decoder with application to image transmission. In *Proc. IEEE Globecom 2001*, pages 1405–1409, San Antonio, TX, November 2001.