Codes convolutifs et codes concaténés associés

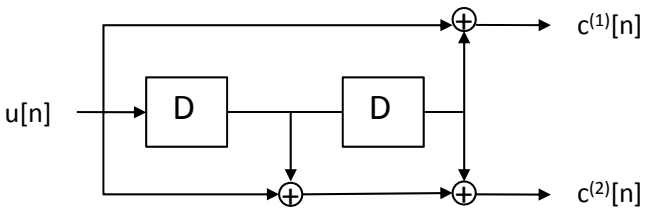
Charly Poulliat

17 octobre 2011

Plan

Introduction : le code $(5, 7)_8$

Représentation par registre à décalage



- Représentation vectorielle : $g^{(1)} = [101] = [5]_8$ et $g^{(2)} = [111] = [7]_8$
- Représentation polynomiale (opérateur du retard D) : $g^{(1)}(D) = 1 + D^2$ et $g^{(2)}(D) = 1 + D + D^2$
- Rendement : $R = k/n = 1/2$ (k entrées, n sorties).
- *Mémoire* $\nu = 2$, *longueur de contrainte* $l_c = \nu + 1$.

Introduction : le code $(5, 7)_8$

Notations

- Relations entrées-sorties en temps : soit $u[n]$ la séquence d'entrée, on a alors comme sorties

$$c^{(i)}[n] = u \circledast g^{(i)}[n], \ \forall i \in \{1, 2\}$$

où $\circledast$ représente le produit de convolution sur $GF(2)$.
- Représentation polynomiale : $c^{(i)}(D) = u(D)g^{(i)}(D), \ \forall i \in \{1, 2\}$
- Représentation vectorielle polynomiale :

$$\underline{c}(D) = [c^{(1)}(D), c^{(2)}(D)] = u(D)[g^{(1)}(D), g^{(2)}(D)] = u(D)G(D)$$

où $G(D)$ matrice génératrice polynômiale de taille $k \times n$.

Introduction : le code $(5, 7)_8$

Terminaison de codage

- Troncature : pas de terminaison particulière du codage. Pour K bits en entrée, on a émis $N = K/R$ bits codés.
- Fermeture de treillis : On ajoute aux K bits d'information ν bits, dits de fermeture, pour rejoindre l'état 'nul' du registre. Cela entraine une baisse du rendement R. Ici, $R = K/2 * (K + 2)$.
- Fermeture circulaire de treillis : tail-biting

Codes convolutifs

Notations

- Rendement : k entrées et n sorties d'où le rendement $R = k/n$.
- Représentation vectorielle polynomiale : soient
  $\underline{u}(D) = [u^{(1)}(D), u^{(2)}(D), \ldots, u^{(k)}(D)]$,
  $\underline{c}(D) = [c^{(1)}(D), c^{(2)}(D), \ldots, c^{(n)}(D)]$

$$\underline{c}(D) = \sum_{i=1}^{k} u^{(i)}(D)\underline{g}_i(D)$$

où $\underline{g}_i(D) = [g_i^{(1)}(D), g_i^{(2)}(D), \ldots g_i^{(n)}(D)]$ et $g_i^{(j)}(D)$ représente le transfert entre l'entrée i et la sortie j.

$$\underline{c}(D) = \underline{u}(D)G(D)$$

où $G(D)$ matrice génératrice polynômiale de taille $k \times n$.

- Matrice de parité : $\underline{c}(D)H^T(D)$ et donc $G(D)H^T(D) = \mathbf{0}$

Codes convolutifs
Notations

- Code récursif : certaines relations entrées sorties peuvent être définies par un code récursif

$$g_i^{(j)}(D) = \frac{b_0 + b_1 D + b_2 D^2 + \ldots + b_m D^m}{1 + a_1 D + a_2 D^2 + \ldots + a_m D^m}$$

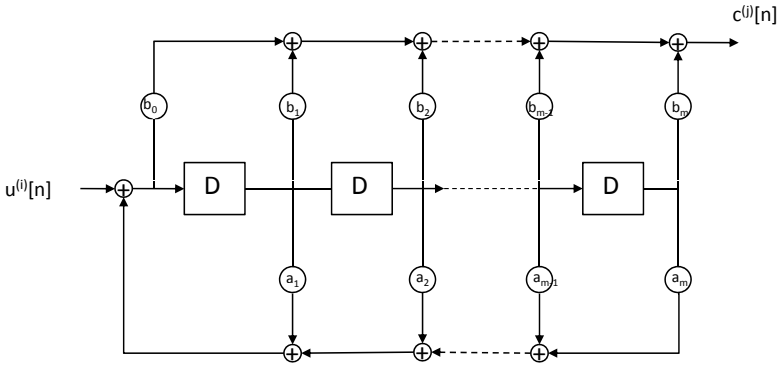


FIGURE: Implémentation d'un transfert récursif par réalisation d'un filtre récursif à réponse impulsionnelle infinie de Type I.

Codes convolutifs
Notations

- Exemple : code récursif systématique $(1, 5/7)_8$ de matrice génératrice

$$G(D) = [1 \; \frac{1 + D^2}{1 + D + D^2}]$$

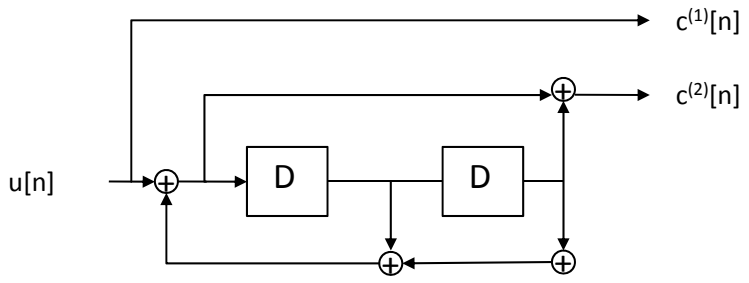
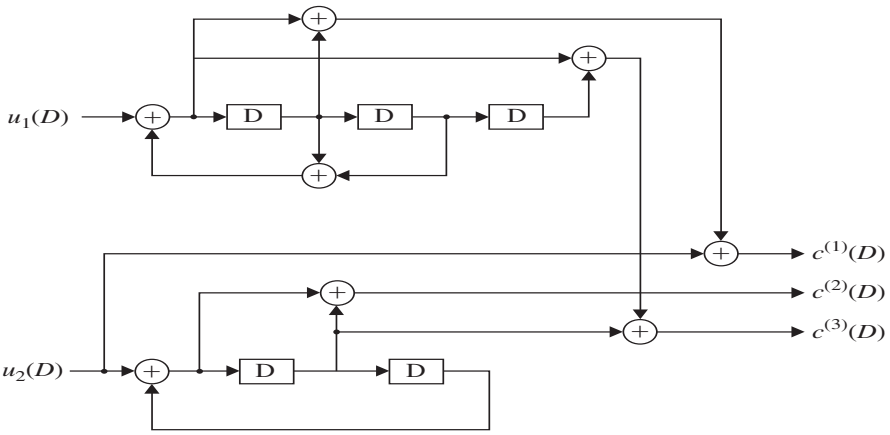


FIGURE: représentation du code récursif (1,5/7)

Codes convolutifs

Notations - exemple pour un code rendement $R = k/n$



$$G(D) = \begin{pmatrix} \frac{1+D}{1+D+D^2} & 0 & 1+D \\ 1 & \frac{1}{1+D} & \frac{D}{1+D^2} \end{pmatrix}$$

$$\underline{u}(D) = [u_1(D), u_2(D)], \underline{c}(D) = [c_1(D), c_2(D), c_3(D)]$$

Codes convolutifs

Modèle d'état et représentation par machine à états finis

- Représentation par machine à états finis : pour un code de type $R = 1/n$,

$$\{u_k\} \rightarrow \boxed{\underline{S}n} \rightarrow \{c_k\}$$

où $c_k = [c_k^{(1)}, c_k^{(2)}, c_k^{(n)}]$ et $\underline{S}_k$ représente l'état interne du registre à décalage.

- Représentation fonctionnelle associée :
  - Equation d'évolution : passage d'un état à $\underline{S}_{k-1}$ à $\underline{S}_k$.

$$\underline{S}_k = F_1(\underline{S}_{k-1}, u_k)$$

  - Equation d'observation : génération des sorties observables c_k.

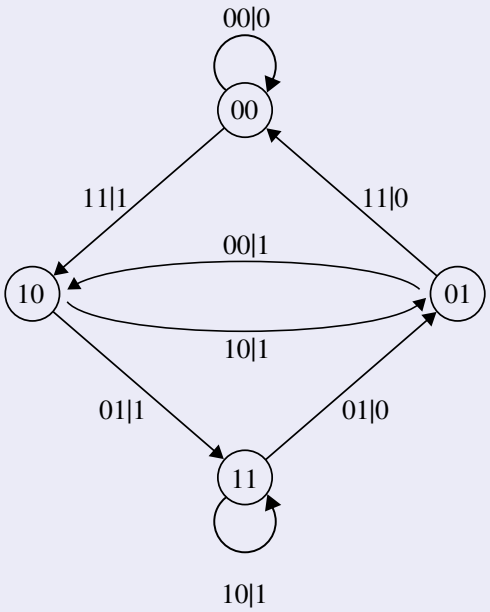
$$c_k = F_2(\underline{S}_{k-1}, u_k)$$

- Exemple : Code $(7, 5)_8$, $\underline{S}_k = [u_k, u_{k-1}]$.

Codes convolutifs

Représentation graphique en Diagramme d'Etat

Code convolutif $(7, 5)_8$



Codes convolutifs

Représentation graphique en treillis

- Definition : représentation graphique du code dans son espace d'état en considérant la dimension temporelle.
  - Noeuds : noeuds du graphe associé à un état $\underline{S}_k$.
  - Transitions : branches entre deux noeuds associées à $F_1(.)$.
  - Etiquettes : informations portées par les branches (u_k, c_k) et données par $F_2(.)$
- Propriétés :
  - Pour un treillis de longueur L, tout chemin du treillis de $\underline{S}_0$ à $\underline{S}_L$ est mot de code obtenu par concaténation des étiquettes c_k du chemin.
  - Le chemin où tous les $\underline{S}_k$ sont l'état **0** (représentation binaire naturelle) est le mot de code nul.
  - *Événement minimal :* chemin qui par de l'état $\underline{S}_k = \mathbf{0}$ et ne revient à cet état que à $\underline{S}_{k+l}$ pour un certain l. On lui associe un poids de Hamming grâce aux étiquettes du chemin.
  - d_{min} est donnée par le poids minimal d'un évênement minimal.

Code convolutif : représentation en treillis

Code $(7, 5)_8$

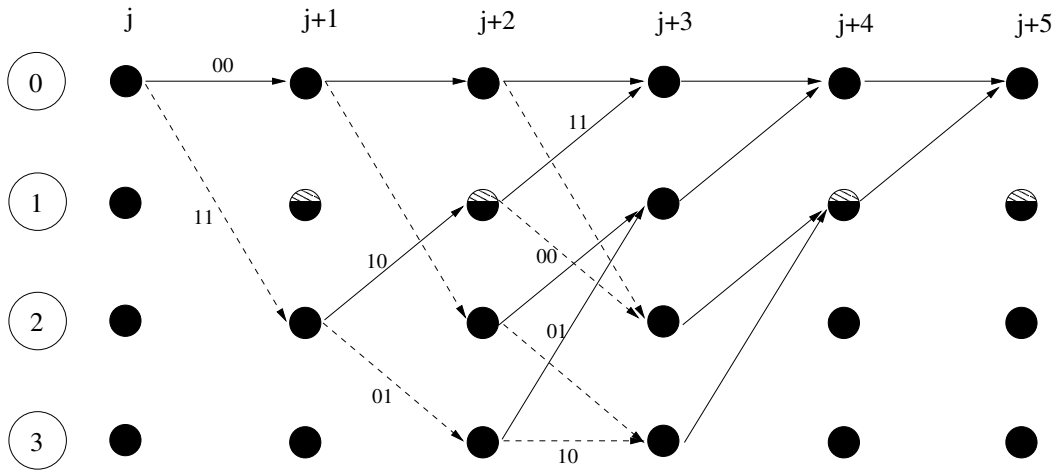


FIGURE: Treillis du code (7,5)

Décodage MAP bit

Notations 1/2

Critère MAP bit

$$\begin{aligned} \hat{u}_n &= \arg \max_{u_n} p(u_n|\mathbf{y}) \\ &= \text{signe}(L(u_n)) \end{aligned} \quad (1)$$

avec

- mapping BPSK : $\{\text{‘0’} \leftrightarrow +1, \text{‘1’} \leftrightarrow -1\}$.
- $u_n \in \{-1, +1\}, \forall n \in [1, L]$
- LLR MAP (Log Likelyhood Ratio) :

$$L(u_n) = \log \left[\frac{p(u_n = +1|\mathbf{y})}{p(u_n = -1|\mathbf{y})} \right]$$

Décodage MAP bit

Notations 2/2

Critère MAP bit

- Longueur de treillis $L = K + Nf$ (K bits d'info. + Nf bits de fermeture).
- Notations vectorielles :

$$\mathbf{c} = [c_1, c_2, \dots c_L]$$

(mot de code émis) avec $c_k = [c_k^{(1)}, c_k^{(2)}, \dots, c_k^{(n_c)}]$

$$\mathbf{y} = [y_1, y_2, \dots y_L]$$

(mot de code reçu) avec $y_k = [y_k^{(1)}, y_k^{(2)}, \dots, y_k^{(n_c)}]$ et $y_k^{(j)} = c_k^{(j)} + b_k^{(j)}$

$$\mathbf{y}_k^l = [y_k, y_{k+1}, \dots y_{l-1}, y_l]$$

Décodage MAP par bit

Algorithme BCJR 1

LLR MAP revisité

$$\begin{aligned} L(u_n) &= \log\left[\frac{p(u_n = +1|\mathbf{y})}{p(u_n = -1|\mathbf{y})}\right] \\ &= \log\left[\frac{\sum_{\mathcal{S}^+} p(s_{n-1} = s', s_n = s, \mathbf{y})}{\sum_{\mathcal{S}^-} p(s_{n-1} = s', s_n = s, \mathbf{y})}\right] \end{aligned} \quad (2)$$

Notations

- Ensemble des transitions associées à $u_n = +1$:

$$\mathcal{S}^+ = \{(s', s) \text{ où } (s_{n-1} = s') \mapsto (s_n = s) | u_n = +1\}$$

- Ensemble des transitions associées à $u_n = -1$:

$$\mathcal{S}^- = \{(s', s) \text{ où } (s_{n-1} = s') \mapsto (s_n = s) | u_n = -1\}$$

Décodage MAP par bit

Algorithme BCJR 2

Factorisation de $p(s', s, \mathbf{y})$

$$p(s_{n-1} = s', s_n = s, \mathbf{y}) = \alpha_{n-1}(s')\gamma_n(s', s)\beta_n(s) \quad (3)$$

$$(4)$$

$$\alpha_n(s) = p(s_n = s, \mathbf{y}_1^n) \quad (5)$$

$$\beta_n(s) = p(\mathbf{y}_{n+1}^L | s_n = s) \quad (6)$$

$$\gamma_n(s', s) = p(s_n = s, y_n | s_{n-1} = s') \quad (7)$$

Récursions forward-backward

$$\alpha_n(s) = \sum_{s'} \gamma_n(s', s)\alpha_{n-1}(s') \quad (8)$$

$$\beta_{n-1}(s') = \sum_{s} \gamma_n(s', s)\beta_n(s) \quad (9)$$

Décodage MAP par bit

Algorithme BCJR 3

Calcul des probabilités de transitions

$$\gamma_n(s', s) = p(s_n = s, y_n | s_{n-1} = s') \quad (10)$$

$$= p(y_n | s', s).p(s | s') \quad (11)$$

$$(12)$$

avec

$$p(s|s') = \begin{cases} 0 & , \text{ si } \{s' \to s\} \text{ non valide} \\ \pi(u_n) & , \text{ sinon} \end{cases}$$

$$\gamma_n(s', s) = p(y_n | c_n(s', s))\pi(u_n)\mathbb{1}_{s' \to s} \quad (13)$$

avec
$\pi(u_n)$ probabilité à priori de u_n
$c_n(s', s)$ les bits associés à l'étiquette $(s' \to s)$

Décodage MAP par bit

Algorithme BCJR 4

Calcul des probabilités de transitions : cas Gaussien

$$y_n^{(m)} = c_n^{(m)} + b_n^{(m)}, b_n^{(m)} \sim \mathcal{N}(0, \sigma_b^2)$$

$$\gamma_n(s', s) \propto \pi(u_n) \exp\left(\frac{\sum_{m=1}^{n_c} |y_n^{(m)} - c_n^{(m)}(s', s)|^2}{2\sigma_b^2}\right) \mathbb{1}_{s' \to s}$$

Initialisation (fermeture treillis)

$$\alpha_0(0) = 1 \quad , \quad \alpha_0(s) = 0 \text{ sinon} \tag{14}$$

$$\beta_L(0) = 1 \quad , \quad \beta_L(s) = 0 \text{ sinon} \tag{15}$$

Décodage MAP par bit

Algorithme BCJR dans domaine logarithmique

Définitions dans le domaine logarithmique

$$\begin{aligned} \tilde{\alpha}_n(s) &\triangleq \log(\alpha_n(s)) \\ &= \log \sum_{s'} \exp\left(\tilde{\alpha}_{n-1}(s') + \tilde{\gamma}_n(s', s)\right) \end{aligned} \tag{16}$$

$$\begin{aligned} \tilde{\beta}_{n-1}(s') &\triangleq \log(\beta_n(s')) \\ &= \log \sum_{s} \exp\left(\tilde{\beta}_n(s) + \tilde{\gamma}_n(s', s)\right) \end{aligned} \tag{17}$$

$$\tilde{\gamma}_n(s', s) \triangleq \log(\gamma_n(s', s)) \tag{18}$$

$$\begin{aligned} L(u_n) &= \log\left(\sum_{\mathcal{S}^+} \exp\left(\tilde{\alpha}_{n-1}(s') + \tilde{\gamma}_n(s', s) + \tilde{\beta}_n(s)\right)\right) \\ &\quad - \log\left(\sum_{\mathcal{S}^-} \exp\left(\tilde{\alpha}_{n-1}(s') + \tilde{\gamma}_n(s', s) + \tilde{\beta}_n(s)\right)\right) \end{aligned} \tag{19}$$

Décodage MAP par bit

Algorithme BCJR dans domaine logarithmique

Opérateur max* (x, y)

$$\max(x, y) = \log\left(\frac{e^x + e^y}{1 + e^{-|x-y|}}\right) \tag{20}$$

$$\begin{aligned} max^*(x, y) &\triangleq \log(e^x + e^y) \\ &= \max(x, y) - \log(1 + e^{-|x-y|}) \end{aligned} \tag{21}$$

$$\begin{aligned} max^*(x, y, z) &\triangleq \log(e^x + e^y + e^z) \\ &= max^*(max^*(x, y), z) \end{aligned} \tag{22}$$

Un exemple : le code $(5, 7)_8$ | Représentation générale | Représentation en treillis | Décodage MAP | Turbo-codes parallèles | Turbo-codes série | Analyse EXIT charts

Décodage MAP par bit

Algorithme BCJR dans domaine logarithmique

Log-MAP (log-BCJR)

$$\tilde{\alpha}_n(s) = \max_{s'}{}^*(\tilde{\alpha}_{n-1}(s') + \tilde{\gamma}_n(s', s)) \tag{23}$$

$$\tilde{\beta}_{n-1}(s') = \max_{s}{}^*(\tilde{\beta}_n(s) + \tilde{\gamma}_n(s', s)) \tag{24}$$

$$\begin{aligned} L(u_n) = &\max_{\mathcal{S}^+}{}^*\left(\tilde{\alpha}_{n-1}(s') + \tilde{\gamma}_n(s', s) + \tilde{\beta}_n(s)\right) \\ &-\max_{\mathcal{S}^-}{}^*\left(\tilde{\alpha}_{n-1}(s') + \tilde{\gamma}_n(s', s) + \tilde{\beta}_n(s)\right) \end{aligned} \tag{25}$$

- Implémentation simplifiée par l'opérateur max(.) et utilisation de lookup table.
- stabilité numérique accrue.

Décodage MAP par bit

Algorithme BCJR dans domaine logarithmique

Log-MAP (log-BCJR) : cas Gaussien

$$\tilde{\gamma}_n(s', s) = \log(\gamma_n(s', s) = -\log(4\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{m=1}^{n_c} |y_n^{(m)} - c_n^{(m)}(s', s)|^2$$

en se réferant au critère MAP final, la constante peut-être supprimée.

Max-Log-MAP

- Remplacer l'opérateur $\max^*(.)$ par l'opérateur $\max(.)$ seul.
- Complexité diminuée, mais perte de performances (raisonnable).
  $\Rightarrow$ décodeur implémenté en pratique

Turbo-codes parallèles

Structure du codeur

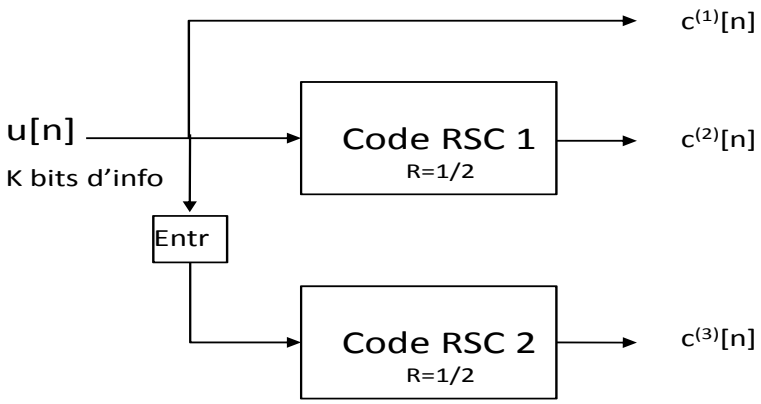


FIGURE: Structure d'un turbo code parallèle : chaque code est un code récursif systématique. Ici $R = 1/2$ pour chaque code constituant.

- K bits d'information sont codés avec le codeur RSC 1,
- les bits d'info. sont entrelacés et codés par le codeur RSC 2,
- Seule la partie redondance est prise en compte sur le codeur RSC 2.

Turbo-codes parallèles

Structure du codeur : exemple UMTS

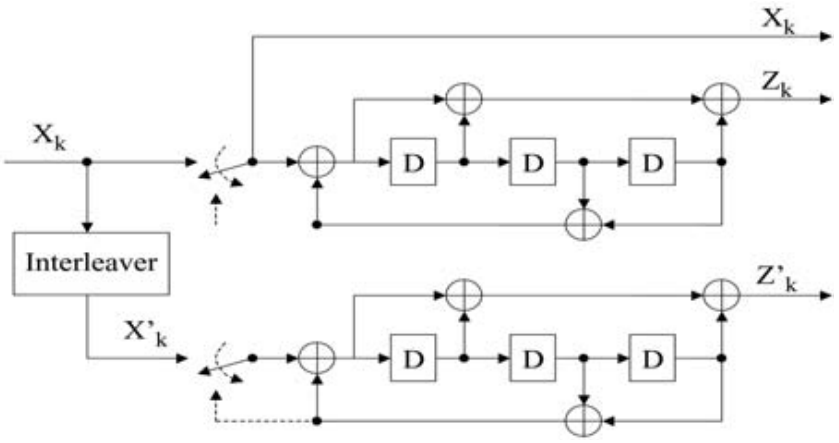


FIGURE: Structure d'un turbo code parallèle pour l'UMTS (3GPP)

Turbo-codes parallèles

Notion d'information extrinsèque 1/3

Cas d'un codeur récursif systématique de rendement $R = 1/n_c$

on supposera que $c_n^{(1)} = u_n$

$$L(u_n) = \log\left[\frac{\sum_{\mathcal{S}^+} p(s_{n-1}=s', s_n=s, \mathbf{y})}{\sum_{\mathcal{S}^-} p(s_{n-1}=s', s_n=s, \mathbf{y})}\right] \quad (26)$$

$$= L_c(u_n) + L_a(u_n) + L_{ext}(u_n) \quad (27)$$

avec

- Info. canal : $L_c(u_n) = \log\left(\frac{p(y_n^{(1)}|u_n=+1)}{p(y_n^{(1)}|u_n=-1)}\right)$
- Info. a priori : $L_a(u_n) = \log\left(\frac{p(u_n=+1)}{p(u_n=-1)}\right)$
- Info. extrinsèque :

$$L_{ext}(u_n) = \log\left[\frac{\sum_{\mathcal{S}^+} \alpha_{n-1}(s') \prod_{k=2}^{n_c} p(y_n^{(k)}|c_n^{(k)}(s',s))\beta_n(s)}{\sum_{\mathcal{S}^-} \alpha_{n-1}(s') \prod_{l=2}^{n_c} p(y_n^{(l)}|c_n^{(l)}(s',s))\beta_n(s)}\right]$$

$$= \log\left[\frac{\sum_{\mathcal{S}^+} \alpha_{n-1}(s')\gamma_n^e(s',s)\beta_n(s)}{\sum_{\mathcal{S}^-} \alpha_{n-1}(s')\gamma_n^e(s',s)\beta_n(s)}\right] \quad (28)$$

Turbo-codes parallèles

Notion d'information extrinsèque 2/3

Info. extrinsèque (domaine logarithmique)

$$\begin{aligned} L_{ext}(u_n) &= \max_{\mathcal{S}^+}{}^{*}\left(\tilde{\alpha}_{n-1}(s') + \tilde{\gamma}_n^e(s', s) + \tilde{\beta}_n(s)\right) \\ &\quad - \max_{\mathcal{S}^-}{}^{*}\left(\tilde{\alpha}_{n-1}(s') + \tilde{\gamma}_n^e(s', s) + \tilde{\beta}_n(s)\right) \end{aligned}$$

$$\tilde{\gamma}_n^e(s', s) = \log\left(\gamma_n^e(s', s)\right)$$

Correspondance entre domaine probabilité vers domaine logarithmique : $L(u_n) \Longleftrightarrow p(u_n)$

$$\begin{aligned} p(u_n) &= \frac{\exp\left(L_a(u_n)/2\right)}{1 + \exp\left(L_a(u_n)\right)} \exp\left(u_n L_a(u_n)/2\right) \\ &= \Lambda_n \exp\left(u_n L_a(u_n)/2\right) \end{aligned} \tag{29}$$

Turbo-codes parallèles

Notion d'information extrinsèque 3/3

Cas Gaussien (domaine logarithmique)

- Info. canal : $L_c(u_n) = \frac{2}{\sigma^2} y_n^{(1)}$,
- Métriques de branches :

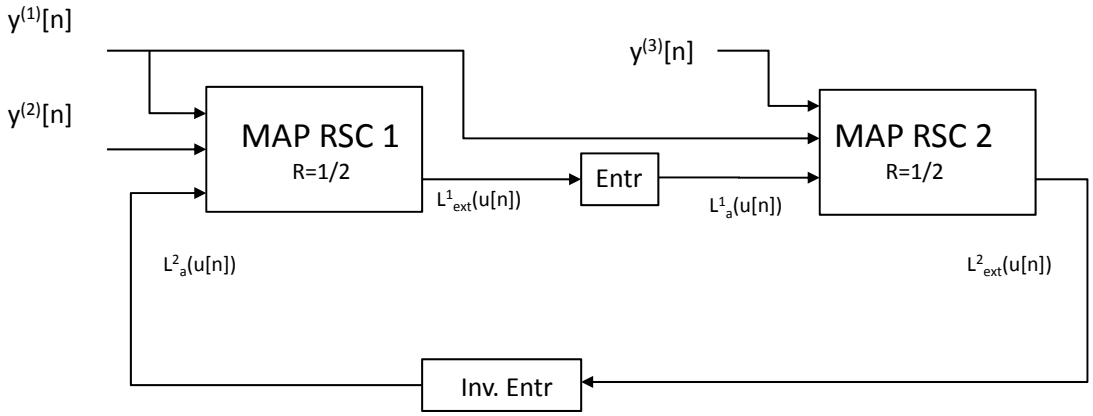
$$\begin{aligned} \tilde{\gamma}_n(s', s) &= u_n \frac{L_a(u_n)}{2} + \frac{1}{2\sigma^2} \sum_{m=1}^{n_c} |y_n^{(m)} - c_n^{(m)}(s', s)|^2 \\ &= u_n \frac{L_a(u_n)}{2} + \sum_{m=1}^{n_c} c_n^{(m)}(s', s) \frac{y_n^{(m)}}{\sigma^2} \\ &= u_n \frac{L_a(u_n)}{2} + \sum_{m=1}^{n_c} c_n^{(m)}(s', s) \frac{L_c(y_n^{(m)})}{2} \end{aligned} \tag{30}$$

$$\tilde{\gamma}_n^e(s', s) = \sum_{m=2}^{n_c} c_n^{(m)}(s', s) \frac{L_c(y_n^{(m)})}{2} \tag{31}$$

$$L(u_n) = \frac{2}{\sigma^2} y_n^{(1)} + L_a(u_n) + L_{ext}(u_n)$$

Turbo-codes parallèles

Décodeur 1/2



- Les décodeurs MAP des deux codes constituants vont s'échanger une information extrinsèque de manière itérative relative aux bits d'information communs.

Turbo-codes parallèles

Décodeur 2/2

- Critères MAP au deux décodeurs à l'itération (ℓ) :

$$\begin{aligned} L^{(\ell)}_{RSC_1}(u_n) &= \frac{2}{\sigma^2} y^{(1)}_n + L^{(\ell-1)}_{a,1}(u_n) + L^{(\ell)}_{e,1}(u_n) \\ &= \frac{2}{\sigma^2} y^{(1)}_n + L^{(\ell-1)}_{e,2}(u_{\pi^{-1}(n)}) + L^{(\ell)}_{e,1}(u_n) \end{aligned} \quad (32)$$

$$L^{(\ell)}_{RSC_2}(u_{\pi(n)}) = \frac{2}{\sigma^2} \pi(y^{(1)}_{\pi(n)}) + L^{(\ell-1)}_{e,1}(u_{\pi(n)}) + L^{(\ell)}_{e,2}(u_{\pi(n)}) \quad (33)$$

avec $\pi(.)$ et π^{-1} représentent les opérations d'entrelacement et de désentrelacement

Turbo-codes parallèles

Performances

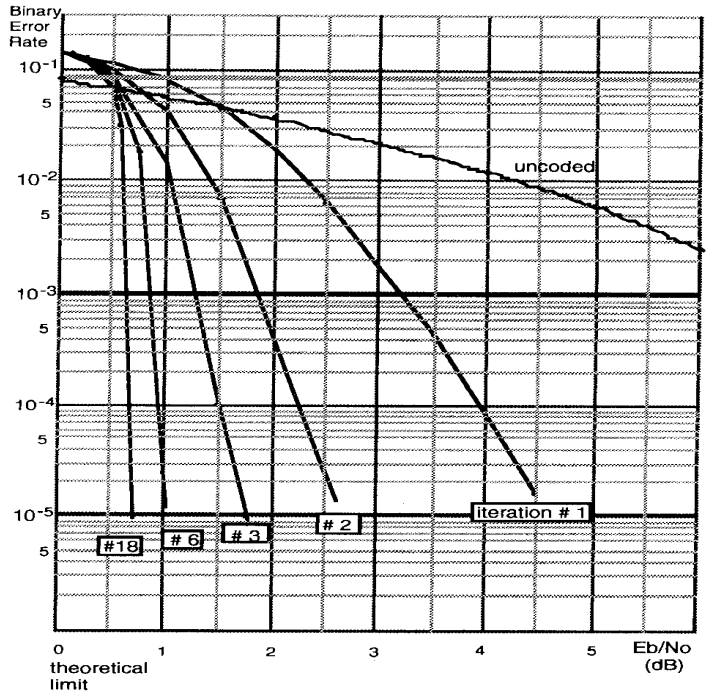
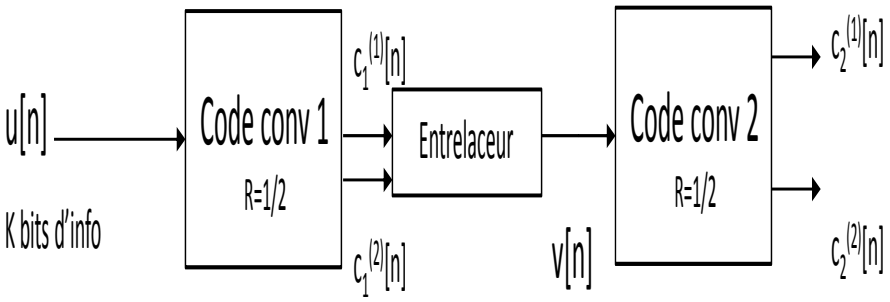


Fig. 9. BER given by iterative decoding ($p = 1, \cdots 18$) of a rate $R = 1/2$ encoder, memory $\nu = 4$, generators $G_1 = 37, G_2 = 21$, with interleaving 256×256.

Turbo-codes série

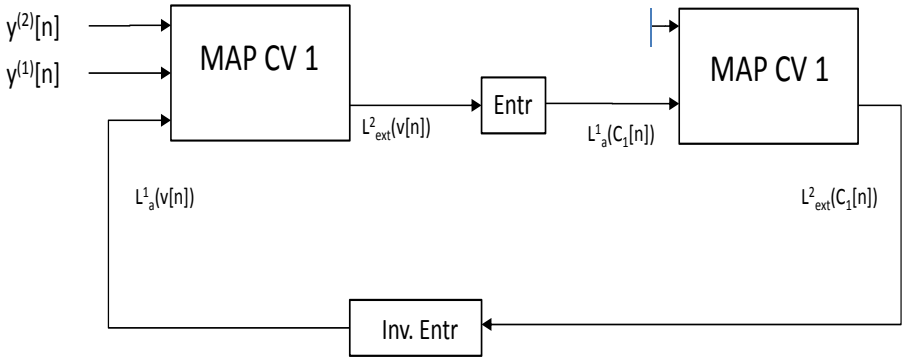
Structure du codeur



- K bits d'information sont codés avec le codeur 1 (code externe),
- les bits codés sont alors entrelacés et puis codés par le codeur 2 (code interne),
- les deux décodeurs associés ne fonctionneront pas forcément de la même manière que pour le cas parallèle,
- seul le codeur 2 doit être récursif pour avoir gain d'entrelacement

Turbo-codes série

Structure du décodeur 1/2



Décodeur code 1

- Idem au décodage d'un bloc du cas parallèle.

Turbo-codes série

Structure du décodeur 2/2

Décodeur code 2

- Critère MAP modifié :

$$
\begin{aligned}
L_1(c_n^{(m)}) &= \max_{\mathcal{C}^+}{}^{*}\left(\tilde{\alpha}_{n-1}(s') + \tilde{\gamma}_n(s', s) + \tilde{\beta}_n(s)\right) \\
&\quad -\max_{\mathcal{C}^-}{}^{*}\left(\tilde{\alpha}_{n-1}(s') + \tilde{\gamma}_n(s', s) + \tilde{\beta}_n(s)\right) \qquad (34) \\
&= L_2^e(c_n^{(m)}) + L_1^e(c_n^{(m)}) \qquad (35) \\
\tilde{\gamma}_n(s', s) &= \sum_{m=1}^{n_c} c_n^{(m)}(s', s)\frac{L_2^e(c_n^{(m)}(s', s))}{2} \qquad (36)
\end{aligned}
$$

avec

$$
\mathcal{C}^+ = \{(s', s) \text{ où } (s_{n-1} = s') \mapsto (s_n = s) | c_n^{(m)} = +1\}
$$

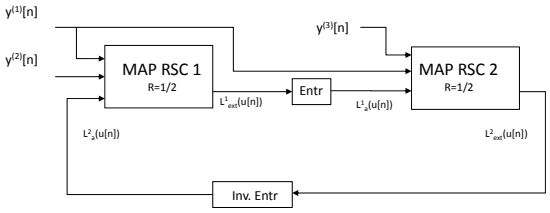
$$
\mathcal{C}^- = \{(s', s) \text{ où } (s_{n-1} = s') \mapsto (s_n = s) | c_n^{(m)} = -1\}
$$

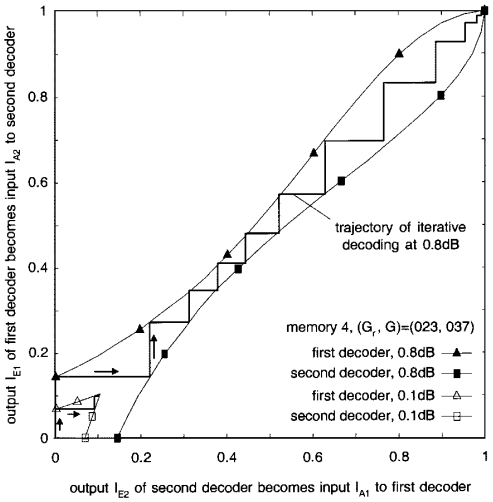
Analyse EXIT charts

Présentation générale

Motivations

- Analyser le comportement d'un turbo-récepteur pour pouvoir prédire les performances en fonction du rapport signal à bruit.





Un exemple : le code $(5, 7)_8$ | Représentation générale | Représentation en treillis | Décodage MAP | Turbo-codes parallèles | Turbo-codes série | Analyse EXIT charts

Analyse EXIT charts

Présentation générale

Idée :

- Pouvoir analyser le comportement entrée-sortie de chaque bloc SISO indépendamment : on cherche à déterminer le comportement moyen en sortie par rapport au comportement moyen en entrée suivant une figure de mérite donnée,
- Ce comportement en entrée est fonction de l'info. extrinsèque entrée et des probabilités de transitions du canal : le canal est connu mais nécessité de modéliser le compotement statistique des informations extrinsèques,
- La figure de mérite en entrée sera associée au info. a priori entrantes (extrinsèques des autres blocs),
- de même, la figure de mérite sortante sera associée aux info extrinsèques fournies par le bloc SISO,
- pour simplifier l'analyse, utiliser une mesure de performance mono-domensionnelle (scalaire) pour caractériser le processus de décodage itératif

Analyse EXIT charts

Présentation générale

EXtrinsic Information Transfer Charts :

- Information mutuelle entre un log-rapport de vraisemblance L et le bit émis C (considérés comme variables aléatoires) :

$$I(L;C) = \frac{1}{2}\sum_{c=\pm 1}\int_{\mathbb{R}} f(l|c)\log_2\left(\frac{2f(l|c)}{f(l|c=+1)+f(l|c=-1)}\right)dl$$

- Hypothèses :
  - Modulation BPSK, entrelacement parfait,
  - C variable binaire de loi uniforme,
  - Symétrie de la densité : $f(l|C=-1) = f(-l|C=+1)$,
  - Consistance de la densité (symétrie exponentielle) : $f(-l|C=c) = f(l|C=c)e^{-c.l}$

$$\Downarrow$$

$$\begin{aligned} I(L;C) &= 1 - \mathbb{E}_{L|C=+1}(\log_2(1+e^{-l})) \\ &= 1 - \int_{\mathbb{R}} f(l|c=+1)\log_2(1+e^{-l})dv \end{aligned}$$

Analyse EXIT charts

Information mutuelle d'une densité consistante : estimation.

Estimation

- Utilisation du mot de code nul :

$$I(L;C) \approx 1 - \frac{1}{N}\sum_n \log_2(1+e^{-l_n})$$

- Mot de codes indifférents et connus :

$$I(L;C) \approx 1 - \frac{1}{N}\sum_n \log_2(1+e^{-c_n l_n})$$

- Mot de codes indifférents et inconnus :

$$I(L;C) \approx 1 - \frac{1}{N}\sum_n \mathcal{H}_b\left(\frac{e^{+|l_n|/2}}{e^{+|l_n|/2}+e^{-|l_n|/2}}\right)$$

$$\mathcal{H}_b(p) = -p\log_2(p) - (1-p)\log_2(1-p)$$

Analyse EXIT charts

Information a priori et extrinsèque

- Modèle Gaussien des messages a priori :

$$I = m.c + b,\ b \sim \mathcal{N}(0, \sigma^2 = 2m),\ c = \pm 1$$

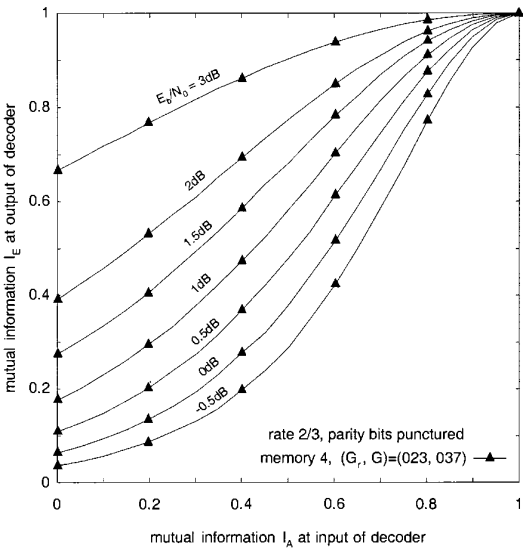
- Information mutuelle associée :

$$I(L; C) = 1 - \frac{1}{\sqrt{2\pi\sigma^2}} \int_{\mathbb{R}} \exp\left(-\frac{(l - \sigma^2/2)^2}{2\sigma^2}\right) \log_2\left(1 + e^{-l}\right) dl$$

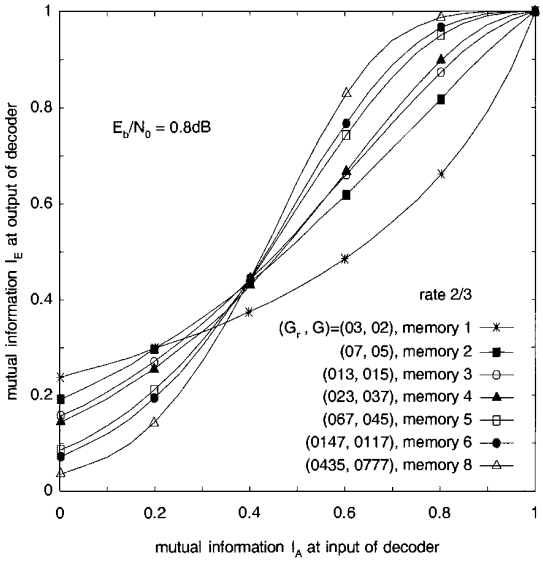
- Information mutuelle extrinsèque : Elle est évaluée **sans approximation Gaussienne** à l'aide des histogrammes des densités en sortie ou à l'aide des estimateurs précédents.

Analyse EXIT charts

Concaténation parallèle 1/3



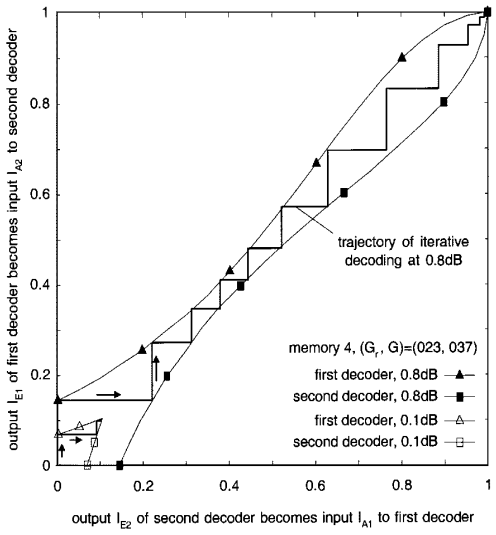
Influence de E_b/N_0

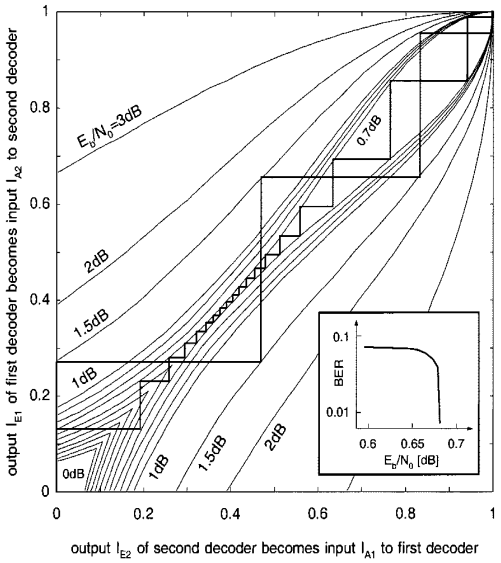


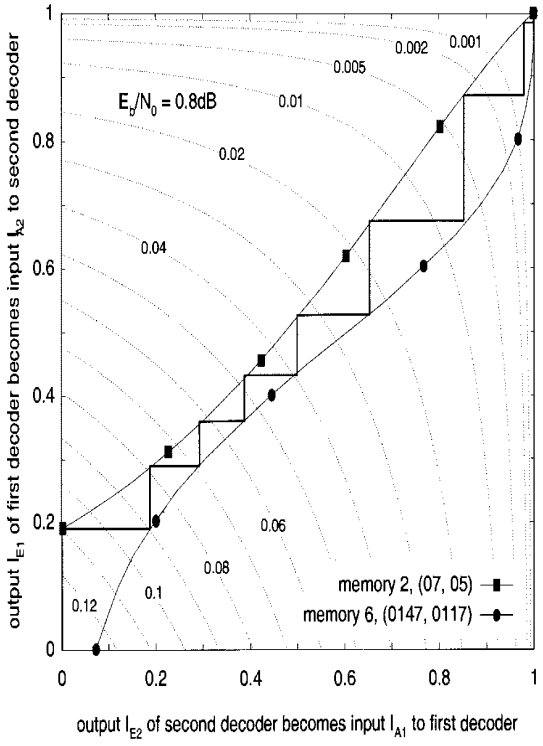
influence de la mémoire

Analyse EXIT charts

Concaténation parallèle 2/3



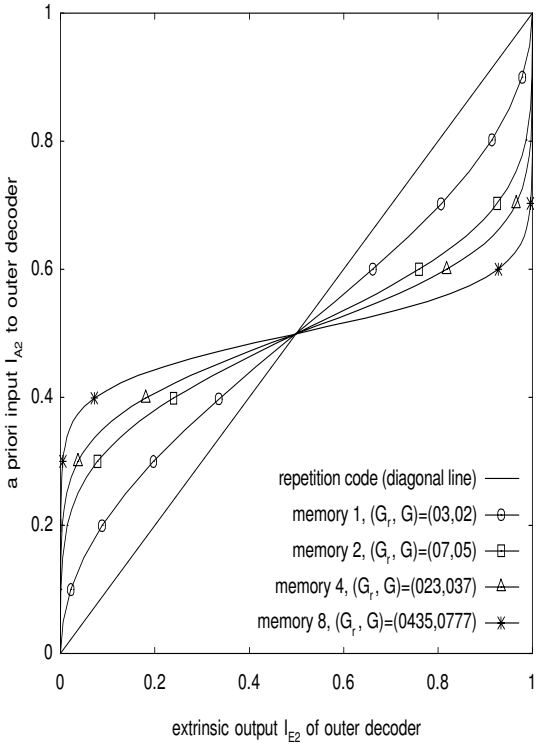


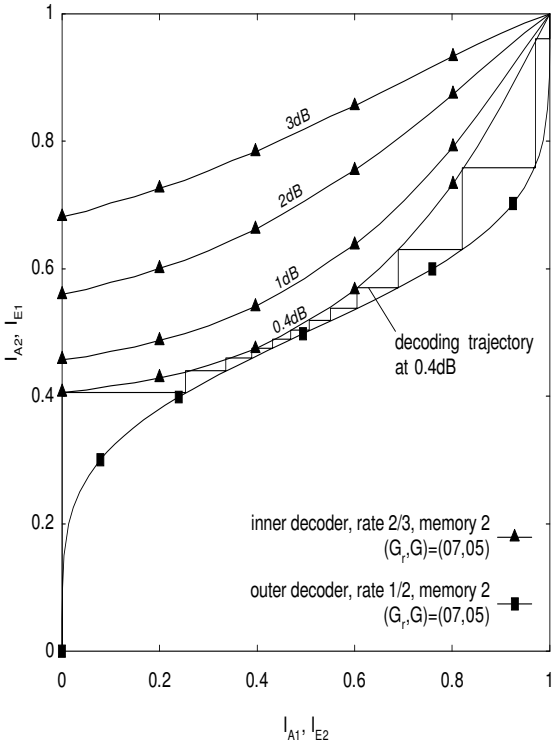


$$P_b \approx erfc\left(\frac{1}{2\sqrt{2}}\sqrt{8R\frac{E_b}{N_0} + J^{-1}(I_A)^2 + J^{-1}(I_E)^2}\right)$$

Analyse EXIT charts

Concaténation série





Bibliographie

- A. Glavieux and all, *Channel coding in communication networks : from theory to turbocodes*, Volume 3 de Digital Signal Image Processing Series, John Wiley Sons, 2007.
- Claude Berrou and all, *Codes and Turbo Codes*, Collection IRIS Series, IRIS International, Springer, 2010.
- W.E. Ryan, Shu Lin, *Channel codes : classical and modern*, Cambridge University Press, 2009.
- Shu Lin, Daniel J. Costello, *Error control coding : fundamentals and applications*, Édition 2, Pearson-Prentice Hall, 2004.
- T. Richardson, R. Urbanke, *Modern coding theory*, Cambridge University Press, 2008.

Concatenated Convolutional Codes and Iterative Decoding

William E. Ryan
Department of Electrical and Computer Engineering
The University of Arizona
Box 210104
Tucson, AZ 85721
ryan@ece.arizona.edu

July 10, 2001

1. Introduction

Turbo codes, first presented to the coding community in 1993 [1], [2], represent one of the most important breakthroughs in coding since Ungerboeck introduced trellis codes in 1982 [3]. A turbo code encoder comprises a concatenation of two (or more) convolutional encoders and its decoder consists of two (or more) "soft" convolutional decoders which feed probabilistic information back and forth to each other in a manner that is reminiscent of a turbo engine. This chapter presents a tutorial exposition of parallel and serial concatenated convolutional codes (PCCCs and SCCCs), which we will also call parallel and serial turbo codes. Included here are a simple derivation for the performance of these codes and a straightforward presentation of their iterative decoding algorithms. The treatment is intended to be a launching point for further study in the field and to provide sufficient information for the design of computer simulations. This chapter borrows from some of the most prominent publications in the field [4]-[12].

The chapter is organized as follows. Section 2 describes details of the parallel and serial turbo code encoders. Section 3 derives a truncated union bound on the error rate of these codes under the assumption of maximum-likelihood decoding. This section explains the phenomenon of interleaver gain attained by these codes. Section 4 derives in detail the iterative (turbo) decoder for both PCCCs and SCCCs. Included in this section are the BCJR decoding algorithm for convolutional codes and soft-in/soft-out decoding modules for use in turbo decoding. The decoding algorithms are presented explicitly to facilitate the creation of computer programs. Section 5 contains a few concluding remarks.

2. Encoder Structures

Fig. 1 depicts a parallel turbo encoder. As seen in the figure, the encoder consists of two binary rate 1/2 convolutional encoders arranged in a so-called parallel concatenation, separated by a K-

bit pseudo-random interleaver or permuter. Also included is an optional puncturing mechanism to obtain high code rates [13]. Clearly, without the puncturer, the encoder is rate 1/3, mapping K data bits to $3K$ code bits. With the puncturer, the code rate $R = K/(K+P)$, where P is the number of parity bits remaining after puncturing. Observe that the constituent encoders are recursive systematic convolutional (RSC) codes. As will be seen in the sequel, recursive encoders are necessary to attain the exceptional performance (attributed to "interleaver gain") provided by turbo codes. Without any essential loss of generality, we assume that the constituent codes are identical.

Fig. 2 depicts a serial turbo encoder. As seen in the figure, the serially concatenated convolutional encoders are separated by an interleaver, and the inner encoder is required to be an RSC code, whereas the outer encoder need not be recursive [6]. However, RSC inner and outer encoders are often preferred since it is convenient to puncture only parity bits to obtain high code rates [14]. Further, an RSC outer code will facilitate our analysis below. The code rate for the serial turbo encoder is $R = R_o \cdot R_i$ where R_o and R_i are the code rates for the outer and inner codes, respectively.

For both parallel and serial turbo codes, the codeword length is $N = K/R$ bits, and we may consider both classes to be (N, K) block codes.

We now discuss in some detail the individual components of the turbo encoders.

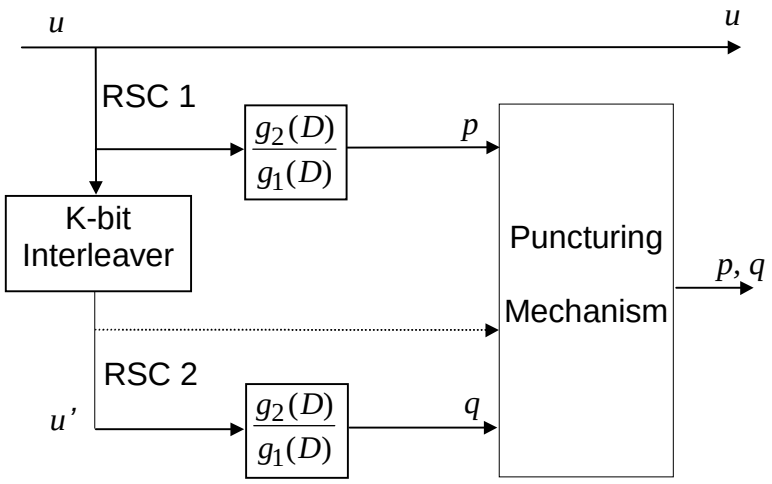


Fig. 1. PCCC encoder diagram.

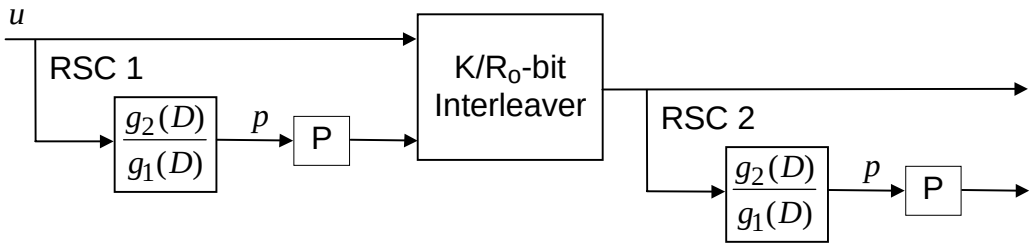


Fig. 2. SCCC encoder diagram with RSC component codes. "P" signifies possible puncturing of parity bits.

2.1. The Recursive Systematic Encoders

Whereas the generator matrix for a rate 1/2 non-recursive convolutional code has the form $G_{NR}(D) = [g_1(D) \quad g_2(D)]$, the equivalent recursive systematic encoder has the generator matrix

$$G_R(D) = \begin{bmatrix} 1 & \dfrac{g_2(D)}{g_1(D)} \end{bmatrix} .$$

Observe that the code sequence corresponding to the encoder input $u(D)$ for the former code is $u(D)G_{NR}(D) = [u(D)g_1(D) \quad u(D)g_2(D)]$, and that the identical code sequence is produced in the recursive code by the sequence $u'(D) = u(D)g_1(D)$, since in this case the code sequence is $u(D)g_1(D)G_R(D) = u(D)G_{NR}(D)$. Here, we loosely call the pair of polynomials $[u(D)g_1(D) \quad u(D)g_2(D)]$ a code sequence, although the actual code sequence is derived from this polynomial pair in the usual way.

Observe that, for the recursive encoder, the code sequence will be of finite weight if and only if the input sequence is divisible by $g_1(D)$. We have the following corollaries of this fact which we shall use later.

Fact 1. A weight-1 input will produce an infinite weight output for such an input is never divisible by a (non-trivial) polynomial $g_1(D)$. (In practice, "infinite" should be replaced by "large" since the input length is finite.)

Fact 2. For any non-trivial $g_1(D)$, there exists a family of weight-2 inputs of the form $D^j(1+D^p)$, $j \geq 0$, which produce finite weight outputs, i.e., which are divisible by $g_1(D)$. When $g_1(D)$ is a primitive polynomial of degree m, then $p = 2^m - 1$. More generally, p is the length of the pseudorandom sequence generated by $g_1(D)$.

Proof: Because the encoder is linear, its output due to a weight-2 input $D^j(1 + D^t)$ is equal to the sum of its outputs due to D^j and D^jD^t. The output due to D^j will be periodic with period p since the encoder is a finite state machine (see Example 1 and Fig. 3): the state at time j must be reached again in a finite number of steps p, after which the state sequence is repeated indefinitely with period p. Now letting $t = p$, the output due to D^jD^p is just the output due to D^j shifted by p bits. Thus, the output due to $D^j(1 + D^p)$ is the sum of the outputs due to D^j and D^jD^p which much be of finite length and weight since all but one period will cancel in the sum. □

In the context of the code's trellis, Fact 1 says that a weight-1 input will create a path that diverges from the all-zeros path, but never remerges. Fact 2 says that there will always exist a trellis path that diverges and remerges later which corresponds to a weight-2 data sequence.

Example 1. Consider the code with generator matrix

$$G_R(D) = \begin{bmatrix} 1 & \dfrac{1 + D^2 + D^3 + D^4}{1 + D + D^4} \end{bmatrix} .$$

Thus, $g_1(D) = 1 + D + D^4$ and $g_2(D) = 1 + D^2 + D^3 + D^4$ or, in octal form, $(g_1, g_2) = (31, 27)$. Observe that $g_1(D)$ is primitive so that, for example, $u(D) = 1 + D^{15}$ produces the finite-length code sequence $(1 + D^{15}, 1 + D + D^2 + D^3 + D^5 + D^7 + D^8 + D^{11})$. Of course, any delayed version of this input, say, $D^7(1 + D^{15})$, will simply produce a delayed version of this code sequence. Fig. 3 gives one encoder realization for this code. We remark that, in addition to elaborating on Fact 2, this example serves to demonstrate the conventions generally used in the literature for specifying such encoders. □

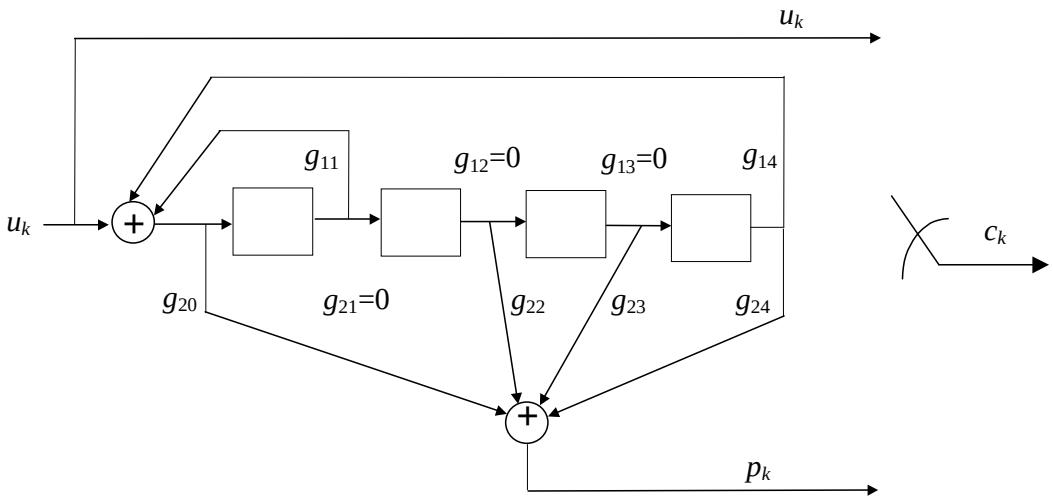


Fig. 3. RSC encoder with $(g_1, g_2) = (31, 27)$.

2.2. The Interleaver

The function of the interleaver is to take each incoming block of bits and rearrange them in a pseudo-random fashion prior to encoding by the second encoder. For the PCCC, the interleaver permutes K bits and, for the SCCC, the interleaver permutes K/R_o bits. Unlike the classical interleaver (e.g., block or convolutional interleaver), which rearranges the bits in some systematic fashion, it is crucial that this interleaver sort the bits in a manner that lacks any apparent order, although it might be tailored in a certain way for weight-2 and weight-3 inputs as will be made clearer below. The $\mathcal{S}$-random interleaver [8] is quite effective in this regard. This particular interleaver ensures that any two inputs bits whose positions are within $\mathcal{S}$ of each other are separated by an amount greater than $\mathcal{S}$ at the interleaver output. $\mathcal{S}$ should be selected to be as large as possible for a given value of K. Also, as we shall see, performance increases with K, and so $K \geq 1000$ is typical.

2.3. The Puncturer

The role of the turbo code puncturer is identical to that of its convolutional code counterpart, that is, to delete selected bits to reduce coding overhead. We have found it most convenient to delete only parity bits, but there is no guarantee that this will maximize the minimum codeword distance. For example, to achieve a rate of 1/2, one might delete all even parity bits from the top encoder and all odd parity bits from the bottom one.

3. Performance with Maximum-Likelihood Decoding

As will be elaborated upon in the next section, a maximum-likelihood (ML) sequence decoder would be far too complex for a turbo code due to the presence of the permuter. However, the suboptimum iterative decoding algorithm to be described there offers near-ML performance. Hence, we shall now estimate the performance of an ML decoder on a binary-input AWGN channel with power spectral density $N_0/2$ (analysis of the iterative decoder is much more difficult).

Armed with the above descriptions of the components of the turbo encoders of Figs. 1 and 2, it is easy to conclude that it is linear since its components are linear. The constituent codes are certainly linear, and the interleaver is linear since it may be modeled by a permutation matrix. Further, the puncturer does not affect linearity since all the constituent codewords share the same puncture locations. As usual, the importance of linearity in evaluating the performance of a code is that one may choose the all-zeros sequence as a reference. Thus, we shall assume that the all-zeros codeword was transmitted. The development below holds for both parallel and serial turbo codes.

Now consider the all-zeros codeword (the 0^{th} codeword) and the k^{th} codeword, for some $k \in \{1, 2, ..., 2^K - 1\}$. The ML decoder will choose the k^{th} codeword over the 0^{th} codeword with probability $Q\left(\sqrt{2d_k RE_b/N_0}\right)$ where d_k is the weight of the k^{th} codeword and E_b is the energy per information bit. The bit error rate for this two-codeword situation would then be

$$\begin{aligned} P_b(k \mid 0) &= w_k \text{ (bit errors/cw error)} \times \\ &\quad \frac{1}{K} \text{ (cw/ data bits)} \times \\ &\quad Q\left(\sqrt{2Rd_k E_b/N_0}\right) \text{ (cw errors/cw)} \\ &= \frac{w_k}{K} Q\left(\sqrt{\frac{2Rd_k E_b}{N_0}}\right) \text{ (bit errors/data bit)} \end{aligned}$$

where w_k is the weight of the k^{th} data word. Now including all of the codewords and invoking the usual union bounding argument, we may write

$$\begin{aligned} P_b &= P_b(\text{choose any } k \in \{1, 2, ..., 2^K - 1\} \mid 0) \\ &\leq \sum_{k=1}^{2^K - 1} P_b(k \mid 0) \\ &= \sum_{k=1}^{2^K - 1} \frac{w_k}{K} Q\left(\sqrt{\frac{2Rd_k E_b}{N_0}}\right) . \end{aligned}$$

Note that every non-zero codeword is included in the above summation. Let us now reorganize the summation as

$$P_b \leq \sum_{w=1}^{K} \sum_{v=1}^{\binom{K}{w}} \frac{w}{K} Q\left(\sqrt{\frac{2Rd_{wv} E_b}{N_0}}\right) \tag{3.1}$$

where the first sum is over the weight-w inputs, the second sum is over the $\binom{K}{w}$ different weight-w inputs, and d_{wv} is the weight of the v^{th} codeword produced by a weight-w input. We emphasize that (3.1) holds for any linear code.

Consider now the first few terms in the outer summation of (3.1) in the context of parallel and serial turbo codes. Analogous to the fact that the top encoder in the parallel scheme is recursive, we shall assume that the outer encoder in the serial scheme is also recursive. By doing so, our arguments below will hold for both configurations. Further, an RSC outer code facilitates the design of high rate serial turbo codes as mentioned above.

$w = 1$: From Fact 1 and associated discussion above, weight-1 inputs will produce only large weight codewords at both PCCC constituent encoder outputs since the trellis paths created never remerge with the all-zeros path. (We ignore the extreme case where the single 1 occurs at the end of the input words for both encoders for this is avoidable by proper interleaver design.) For the SCCC, the output of the outer encoder will have large weight due to Fact 1, and its inner encoder output will have large weight since its input has large weight. Thus, for both cases, each d_{1v} can be expected to be significantly greater than the minimum codeword weight so that the $w = 1$ terms in (3.1) will be negligible.

$w = 2$: Suppose that of the $\binom{K}{2}$ possible weight-2 encoder inputs, $u_*(D)$ is the one of least degree that yields the minimum turbo codeword weight, $d_{2,\min}$, for weight-2 inputs. Due to the presence of the pseudo-random interleaver, the encoder is not time-invariant, and only a small fraction of the inputs of the form $D^j u_*(D)$ (there are approximately K of them) will also produce turbo codewords of weight $d_{2,\min}$. (This phenomenon has been called *spectral thinning* [10].) Denoting by n_2 the number of weight-2 inputs which produce weight-$d_{2,\min}$ turbo codewords, we may conclude that $n_2 << K$. (For comparison, $n_2 \simeq K$ for a single RSC code as shifts of some worst-case input merely shifts the encoder output, thus maintaining a constant output weight.) Further, the overall minimum codeword weight, $d_{\min}$, is likely to be equal or close to $d_{2,\min}$ since low-degree, low-weight input words tend to produce low-weight codewords. (This is easiest to see in the parallel turbo code case which is systematic.)

$w \geq 3$: When w is small (e.g., $w = 3$ or 4), an argument similar to the $w = 2$ case may be made to conclude that the number of weight-w inputs, n_w, that yield the minimum turbo codeword weight for weight-w inputs, $d_{w,\min}$, is such that $n_w << K$. Further, we can expect $d_{w,\min}$ to be equal or close to $d_{\min}$. No such arguments can be made as w increases beyond about 5.

To summarize, by preserving only the dominant terms, the bound in (3.1) can be approximated as

$$P_b \simeq \sum_{w=2}^{3} \frac{w n_w}{K} Q\left(\sqrt{\frac{2Rd_{w,\min}E_b}{N_0}}\right) \tag{3.2}$$

where $w \geq 4$ terms may be added in the event that they are not negligible (more likely to be necessary for SCCCs). We note that n_w and $d_{w,\min}$ are functions of the particular interleaver employed. Since $w = 2$ or 3 in (3.2) and $n_w << K$ with $K \geq 1000$, the coefficients out in front of the Q-function are much less than unity. (For comparison, the coefficient for a convolutional code can be much greater than unity, cf. [10]) This effect is called *interleaver gain* and demonstrates the necessity of large interleavers. Finally, we note that recursive encoders are crucial elements of a turbo code since, for non-recursive encoders, division by $g_1(D)$ (non-remergent trellis paths) would not be an issue and (3.2) would not hold (although (3.1) still would).

When $K \simeq 1000$, it is possible to exhaustively find via computer the weight spectra $\{d_{2v} : v = 1, ..., \binom{K}{2}\}$ and $\{d_{3v} : v = 1, ..., \binom{K}{3}\}$ corresponding to the weight-2 and -3 inputs. In this case, an improved estimate of P_b, given by a truncation of (3.1), is

$$P_b \simeq \sum_{w=2}^{3} \sum_{v=1}^{\binom{K}{w}} \frac{w}{K} Q\left(\sqrt{\frac{2Rd_{wv}E_b}{N_0}}\right). \tag{3.3}$$

We remark that if codeword error rate, P_{cw}, is the preferred performance metric, then an

estimate of P_{cw} may be obtained from (3.2) or (3.3) by removing the factor w/K from these expressions. That this is so may be seen by following the derivation above for P_b.

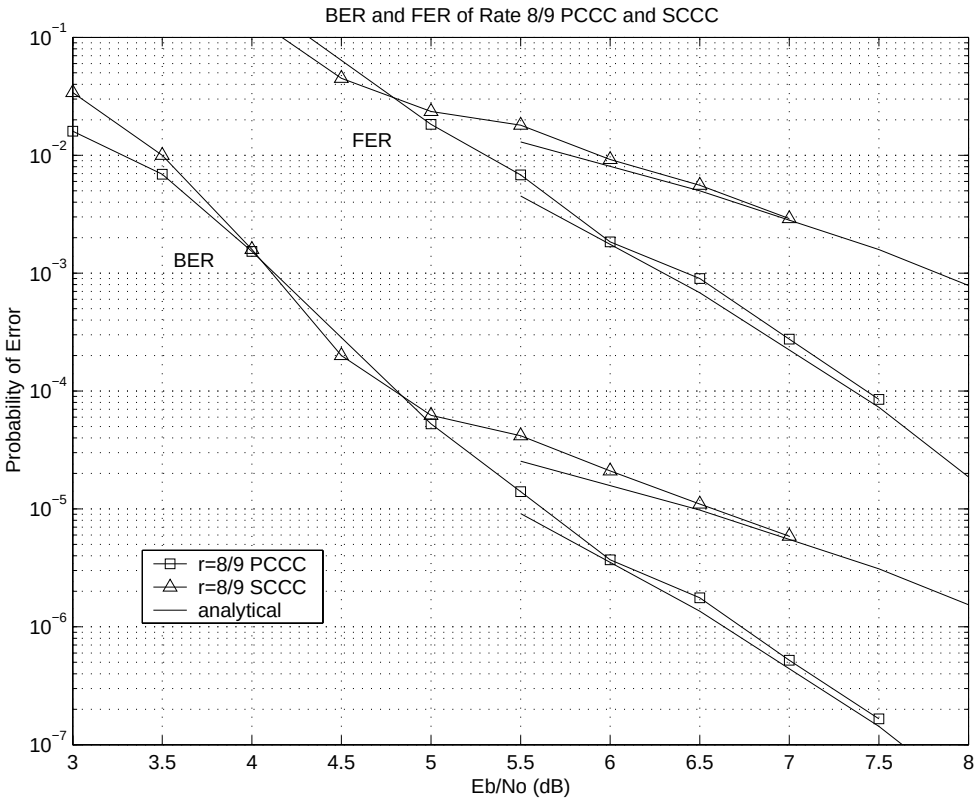


Fig. 4. PCCC and SCCC bit error rate (BER) and frame error rate (FER) simulation results together with analytical result in (3.3).

Example 2. We consider in this example a PCCC and an SCCC code, both rate 8/9 with parameters $(N, K) = (1152, 1024)$. We use identical 4-state RSC encoders in the PCCC encoder whose generators polynomials are, in octal form, $(g_1, g_2) = (7, 5)$. To achieve a code rate of 8/9, only one bit is saved in every 16-bit block of parity bits at each encoder output. The outer constituent encoder in the SCCC encoder is this same 4-state RSC encoder, and the inner code is a rate-1 differential encoder with transfer function $\frac{1}{1 \oplus D}$. A rate of 8/9 is achieved in this case by saving one bit in every 8-bit block of parity bits. The PCCC interleaver is a 1024-bit pseudo-random interleaver with no constraints added (e.g., no $\mathcal{S}$-random constraint). The SCCC interleaver is a 1152-bit pseudo-random interleaver with no constraints added.

Fig. 4 presents performance results for these codes based on computer simulation using the iterative (i.e., non-ML) decoding algorithm of the next section. Simulation results for both bit error rate P_b (BER in the figure) and frame or codeword error rate P_{cw} (FER in the figure) are presented. Also included in the figure are analytic performance curves for ML decoding using the truncated union bound in (3.3). (P_{cw} is obtained by removing the factor w/K in (3.3) as indicated above.) We see the close agreement between the analytical and simulated results in this figure. □

In addition to illustrating the use of the estimate (3.3), this example helps explain the "flooring" effect of the error rate curves: it may be interpreted as the usual Q-function shape

for a signaling scheme with a modest $d_{\min}$, "pushed down" by the interleaver gain $w^* n_{w^*}/K$, where w^* is the value of w corresponding to the dominant term in (3.2) or (3.3).

We comment on the fact that the PCCC in Fig. 4 is substantially better than the SCCC whereas it is known that SCCCs generally have lower floors [6]. We attribute this to the fact that the outer RSC code in the SCCC has been punctured so severely that $d_{\min} = 2$ for this outer code (although $d_{\min}$ for the SCCC is a bit larger). The RSC encoders for the PCCC is punctured only half as much, and so $d_{\min} > 2$ for each of these encoders. We also attribute this to the fact that we have not used an optimized interleaver for this example. In support of these comments, we have also simulated rate 1/2 versions of this same code structure so that no puncturing occurs for the SCCC and much less occurs for the PCCC. In this case, $(N, K) = (2048, 1024)$ and $\mathcal{S}$-random interleavers were used ($\mathcal{S} = 16$ for PCCC and $\mathcal{S} = 20$ for SCCC). The results are presented in Fig. 5 where we observe that the SCCC has a much lower error rate floor, particularly for the FER curves. Finally, we remark that $w \geq 4$ terms in (3.2) are necessary for an accurate estimate of the floor level of the SCCC case in Fig. 5.

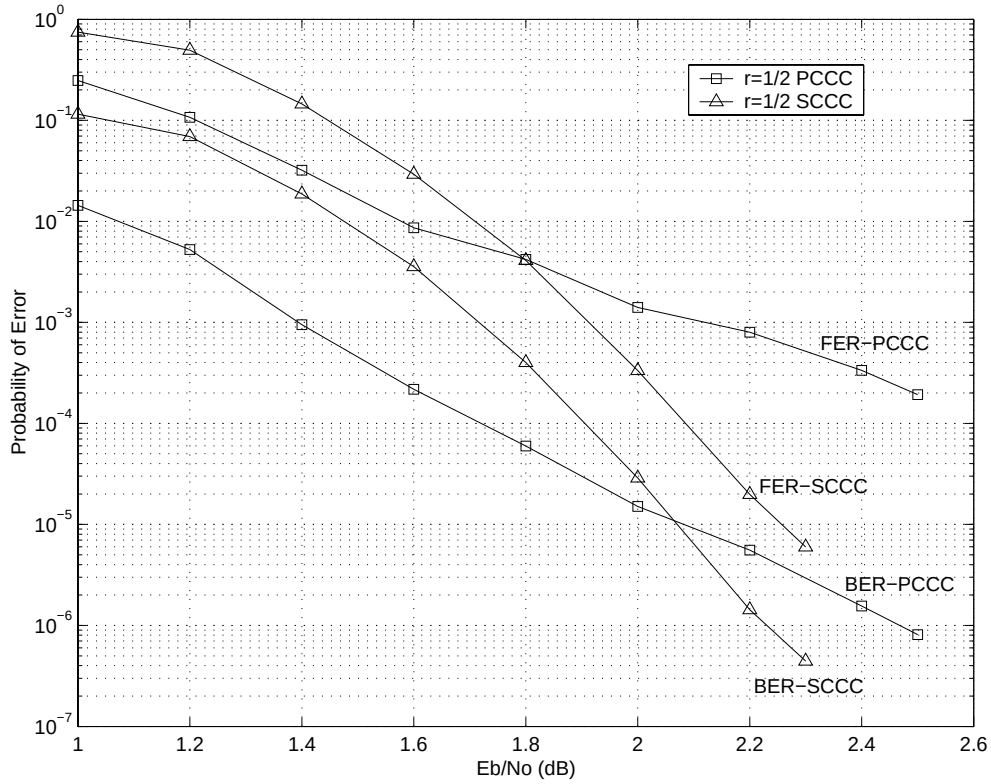


Fig. 5. Rate 1/2 PCCC and SCCC bit error rate (BER) and frame error rate (FER) simulation results.

4. The Iterative Decoders

4.1. Overview of the Iterative Decoder

Consider first an ML decoder for a rate 1/2 convolutional code (recursive or not), and assume a data word of length $K \geq 1000$. Ignoring the structure of the code, a naive ML decoder would

have to compare (correlate) 2^K code sequences to the noisy received sequence, choosing in favor of the codeword with the best correlation metric. Clearly, the complexity of such an algorithm is exorbitant. Fortunately, as we know, such a brute force approach is simplified greatly by Viterbi's algorithm which permits a systematic elimination of candidate code sequences.

Unfortunately, we have no such luck with turbo codes, for the presence of the interleaver immensely complicates the structure of a turbo code trellis. A near-optimal solution is an iterative decoder (also called a turbo decoder) involving two soft-in/soft-out (SISO) decoders which share probabilistic information cooperatively and iteratively. The goal of the iterative decoder is to iteratively estimate the *a posteriori* probabilities (APPs) $\Pr(u_k \mid \mathbf{y})$ where u_k is the k^{th} data bit, $k = 1, 2, ..., K$, and $\mathbf{y}$ is the received codeword in noise, $\mathbf{y} = \mathbf{c} + \mathbf{n}$. In this equation, we assume the components of $\mathbf{c}$ take values in the set $\{\pm 1\}$ (and similarly for $\mathbf{u}$) and $\mathbf{n}$ is a noise word whose components are AWGN samples. Knowledge of the APPs allows for optimal decisions on the bits u_k via the maximum *a posteriori* (MAP) rule[1]

$$\frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})} \underset{-1}{\overset{+1}{\gtrless}} 1$$

or, more conveniently,

$$\hat{u}_k = \text{sign } [L(u_k)] \tag{4.1}$$

where $L(u_k)$ is the log *a posteriori* probability (log-APP) ratio defined as

$$L(u_k) \triangleq \log \left(\frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})} \right) .$$

We shall use the term log-likelihood ratio (LLR) in place of log-APP ratio for consistency with the literature.

The component SISO decoders which jointly estimate the LLRs $L(u_k)$ for parallel and serial turbo codes compute the LLRs for component code inputs (u_{ik}), component code outputs (c_{ik}), or both. Details on the SISO decoders will be presented below. For now, we simply introduce the convention that, for PCCCs, the top component encoder is encoder 1 (denoted E1) and the bottom component decoder is encoder 2 (denoted E2). For SCCCs, the outer encoder is encoder 1 (E1) and the inner encoder is encoder 2 (E2). The SISO component decoders matched to E1 and E2 will be denoted by D1 and D2, respectively. Because the SISO decoders D1 and D2 compute $L(u_{ik})$ and/or $L(c_{ik})$, $i = 1, 2$, we will temporarily use the notation $L(b_k)$ where b_k represents either u_{ik} or c_{ik}.

¿From Bayes' rule, the LLR for an arbitrary SISO decoder can be written as

$$L(b_k) = \log \left(\frac{P(\mathbf{y} \mid b_k = +1)}{P(\mathbf{y} \mid b_k = -1)} \right) + \log \left(\frac{P(b_k = +1)}{P(b_k = -1)} \right) \tag{4.2}$$

with the second term representing *a priori* information. Since $P(b_k = +1) = P(b_k = -1)$ typically, the *a priori* term is usually zero for conventional decoders. However, for *iterative* decoders, each component decoder receives *extrinsic* or *soft* information for each b_k from its companion

[1]It is well known that the MAP rule minimizes the probability of bit error. For comparison, the ML rule, which maximizes the likelihoods $P(\mathbf{y} \mid \mathbf{c})$ over the codewords $\mathbf{c}$, minimizes the probability of codeword error.

decoder which serves as *a priori* information. The idea behind extrinsic information is that D2 provides soft information to D1 for each b_k using only information not available to D1, and D1 does likewise for D2. For SCCCs, the iterative decoding proceeds as D2→D1→D2→D1→..., with the previous decoder passing soft information along to the next decoder at each half-iteration. For PCCCs, either decoder may initiate the chain of component decodings or, for hardware implementations, D1 and D2 may operate simultaneously.

This type of iterative algorithm is known to converge to the true value of the LLR $L(u_k)$ for the concatenated code provided that the graphical representation of this code contains no loops [15]-[17]. The graph of a turbo code does in fact contain loops [17], but the algorithm nevertheless provides near-optimal performance for virtually all turbo codes. That this is possible even in the presence of loops is not fully understood.

This section provided an overview of the turbo decoding algorithm in part to motivate the next section on SISO decoding of a single RSC code using the BCJR algorithm [18]. Following the description of the SISO decoder for a single RSC code will be sections that describe in full detail the iterative PCCC and SCCC decoders which utilize slightly modified SISO decoders.

4.2. The BCJR Algorithm and SISO Decoding

4.2.1. Probability Domain BCJR Algorithm for RSC Codes

Before we discuss the BCJR algorithm in the context of a turbo code, it is helpful to first consider the BCJR algorithm applied to a single rate 1/2 RSC code on an AWGN channel. Thus, as indicated in Fig. 3, the transmitted codeword $\mathbf{c}$ will have the form $\mathbf{c} = [c_1, c_2, ..., c_K] = [u_1, p_1, u_2, p_2, ..., u_K, p_K]$ where $c_k \triangleq [u_k, p_k]$. The received word $\mathbf{y} = \mathbf{c} + \mathbf{n}$ will have the form $\mathbf{y} = [y_1, y_2, ..., y_K] = [y_1^u, y_1^p, y_2^u, y_2^p, ..., y_K^u, y_K^p]$ where $y_k \triangleq [y_k^u, y_k^p]$, and similarly for $\mathbf{n}$. As above, we assume our binary variables take values from the set $\{\pm 1\}$.

Our goal is the development of the BCJR algorithm [18] for computing the LLR

$$L(u_k) = \log\left(\frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})}\right)$$

given the received word $\mathbf{y}$. In order to incorporate the RSC code trellis into this computation, we rewrite $L(u_k)$ as

$$L(u_k) = \log \frac{\sum\limits_{U+} p\left(s_{k-1} = s', s_k = s, \mathbf{y}\right)}{\sum\limits_{U^-} p\left(s_{k-1} = s', s_k = s, \mathbf{y}\right)} \tag{4.3}$$

where s_k is encoder state at time k, U^+ is set of pairs (s', s) for the state transitions $(s_{k-1} = s') \rightarrow (s_k = s)$ which correspond to the event $u_k = +1$, and U^- is similarly defined. To write (4.3) we used Bayes' rule, total probability, and then cancelled $1/p\left(\mathbf{y}\right)$ in the numerator and denominator. We see from (4.3) that we need only compute $p\left(s', s, \mathbf{y}\right) = p\left(s_{k-1} = s', s_k = s, \mathbf{y}\right)$ for all state transitions and then sum over the appropriate transitions in the numerator and denominator. We now provide the crucial results which facilitate the computation of $p\left(s', s, \mathbf{y}\right)$.

Result 1. The pdf $p\left(s', s, \mathbf{y}\right)$ may be factored as

$$p\left(s', s, \mathbf{y}\right) = \alpha_{k-1}\left(s'\right) \cdot \gamma_k\left(s', s\right) \cdot \beta_k\left(s\right) \tag{4.4}$$

where

$$\alpha_k(s) \triangleq p\left(s_k = s, \mathbf{y}_1^k\right)$$

$$\gamma_k(s', s) \triangleq p(s_k = s, y_k \mid s_{k-1} = s')$$

$$\beta_k(s) \triangleq p\left(\mathbf{y}_{k+1}^K \mid s_k = s\right)$$

and where $\mathbf{y}_a^b \triangleq [y_a, y_{a+1}, \cdots, y_b]$.

Proof: By several applications of Bayes' rule, we have

$$\begin{aligned}
p(s', s, \mathbf{y}) &= p\left(s', s, \mathbf{y}_1^{k-1}, y_k, \mathbf{y}_{k+1}^K\right) \\
&= p\left(\mathbf{y}_{k+1}^K \mid s', s, \mathbf{y}_1^{k-1}, y_k\right) p\left(s', s, \mathbf{y}_1^{k-1}, y_k\right) \\
&= p\left(\mathbf{y}_{k+1}^K \mid s', s, \mathbf{y}_1^{k-1}, y_k\right) p\left(s, y_k \mid s', \mathbf{y}_1^{k-1}\right) \cdot p\left(s', \mathbf{y}_1^{k-1}\right) \\
&= p\left(\mathbf{y}_{k+1}^K \mid s\right) \cdot p(s, y_k \mid s') \cdot p\left(s', \mathbf{y}_1^{k-1}\right) \\
&= \beta_k(s) \cdot \gamma_k(s', s) \cdot \alpha_{k-1}(s')
\end{aligned}$$

where the fourth line follows from the third because the variables omitted on the fourth line are conditionally independent. □

Result 2. The probability $\alpha_k(s)$ may be computed in a "forward recursion" via

$$\alpha_k(s) = \sum_{s'} \gamma_k(s', s)\, \alpha_{k-1}(s') \tag{4.5}$$

where the sum is over all possible encoder states.

Proof: By several applications of Bayes' rule and the theorem on total probability, we have

$$\begin{aligned}
\alpha_k(s) &\triangleq p\left(s, \mathbf{y}_1^k\right) \\
&= \sum_{s'} p\left(s', s, \mathbf{y}_1^k\right) \\
&= \sum_{s'} p\left(s, y_k \mid s', \mathbf{y}_1^{k-1}\right) p\left(s', \mathbf{y}_1^{k-1}\right) \\
&= \sum_{s'} p(s, y_k \mid s')\, p\left(s', \mathbf{y}_1^{k-1}\right) \\
&= \sum_{s'} \gamma_k(s', s)\, \alpha_{k-1}(s')
\end{aligned}$$

where the fourth line follows from the third due to conditional independence of $\mathbf{y}_1^{k-1}$. □

Result 3. The probability $\beta_k(s)$ may be computed in a "backward recursion" via

$$\beta_{k-1}(s') = \sum_{s} \beta_k(s)\, \gamma_k(s', s) \tag{4.6}$$

Proof: Applying Bayes' rule and the theorem on total probability, we have

$$\beta_{k-1}(s') \triangleq p\left(\mathbf{y}_k^K \mid s'\right)$$

$$
\begin{aligned}
&= \sum_s p\left(\mathbf{y}_k^K, s \mid s'\right) \\
&= \sum_s p\ \left(\mathbf{y}_{k+1}^K \mid s', s, y_k\right) p\left(s, y_k \mid s'\right) \\
&= \sum_s p\left(\mathbf{y}_{k+1}^K \mid s\right) p\left(s, y_k \mid s'\right) \\
&= \sum_s \beta_k(s)\, \gamma_k\left(s', s\right)
\end{aligned}
$$

where conditional independence led to the omission of variables on the fourth line. □

The recursion for the $\{\alpha_k(s)\}$ is initialized according to

$$
\alpha_0(s) = \begin{cases} 1, & s = 0 \\ 0, & s \neq 0 \end{cases}
$$

which makes the reasonable assumption that the convolutional encoder is initialized to the zero state. The recursion for the $\{\beta_k(s)\}$ is initialized according to

$$
\beta_K(s) = \begin{cases} 1, & s = 0 \\ 0, & s \neq 0 \end{cases}
$$

which assumes that "termination bits" have been appended at the end of the data word so that the convolutional encoder is again in state zero at time K.

All that remains at this point is the computation of $\gamma_k\left(s', s\right) = p\left(s, y_k \mid s'\right)$. Observe that $\gamma_k\left(s', s\right)$ may be written as

$$
\begin{aligned}
\gamma_k\left(s', s\right) &= \frac{P\left(s', s\right)}{P\left(s'\right)} \cdot \frac{p\left(s', s, y_k\right)}{P\left(s', s\right)} \\
&= P\left(s \mid s'\right) p\left(y_k \mid s', s\right) \\
&= P\left(u_k\right) p\left(y_k \mid u_k\right) \qquad (4.7)
\end{aligned}
$$

where the event u_k corresponds to the event $s' \rightarrow s$. Note $P\left(s \mid s'\right) = P\left(s' \rightarrow s\right) = 0$ if s is not a valid state from state s' and $P\left(s' \rightarrow s\right) = 1/2$ otherwise (since we assume binary-input encoders with equiprobable inputs). Hence, $\gamma_k\left(s', s\right) = 0$ if $s' \rightarrow s$ is not valid and, otherwise,

$$
\begin{aligned}
\gamma_k(s', s) &= \frac{P\left(u_k\right)}{\sqrt{2\pi}\sigma} \exp\left[-\frac{\left\|y_k - c_k\right\|^2}{2\sigma^2}\right] \qquad (4.8) \\
&= \frac{1}{2\sqrt{2\pi}\sigma} \exp\left[-\frac{\left(y_k^u - u_k\right)^2 + \left(y_k^p - p_k\right)^2}{2\sigma^2}\right] \qquad (4.9)
\end{aligned}
$$

where $\sigma^2 = N_0/2$.

In summary, we may compute $L(u_k)$ via (4.3) using (4.4), (4.5), (4.6), and (4.8). This "probability domain" version of the BCJR algorithm is numerically unstable for long and even moderate codeword lengths, and so we now present the stable "log domain" version of it.

4.2.2. Log Domain BCJR Algorithm for RSC Codes

In the log-BCJR algorithm, $\alpha_k(s)$ is replaced by the *forward metric*

$$\begin{aligned}\tilde{\alpha}_k(s) &\triangleq \log(\alpha_k(s)) \\ &= \log\left(\sum_{s'} \alpha_{k-1}(s')\,\gamma_k(s',s)\right) \\ &= \log\left(\sum_{s'} \exp\left(\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s',s)\right)\right) \end{aligned} \tag{4.10}$$

where the *branch metric* $\tilde{\gamma}_k(s',s)$ is given by

$$\begin{aligned}\tilde{\gamma}_k(s',s) &= \log\gamma_k(s',s) \\ &= -\log\left(2\sqrt{2\pi}\sigma\right) - \frac{\|y_k - c_k\|^2}{2\sigma^2}. \end{aligned} \tag{4.11}$$

We will see that the first term in (4.11) may be dropped. Note that (4.10) not only defines $\tilde{\alpha}_k(s)$, but it gives its recursion. These log-domain forward metrics are initialized as

$$\tilde{\alpha}_0(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases} \tag{4.12}$$

The probability $\beta_{k-1}(s')$ is replaced by the *backward metric*

$$\begin{aligned}\tilde{\beta}_{k-1}(s') &\triangleq \log(\beta_{k-1}(s')) \\ &= \log\left(\sum_{s} \exp\tilde{\beta}_k(s) + \tilde{\gamma}_k(s',s)\right) \end{aligned} \tag{4.13}$$

with initial conditions

$$\tilde{\beta}_K(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases} \tag{4.14}$$

under the assumption that the encoder has been terminated.

As before, the $L(u_k)$ is computed as

$$\begin{aligned}L(u_k) &= \log\frac{\sum\limits_{U^+} \alpha_{k-1}(s')\,\gamma_k(s',s)\,\beta_k(s)}{\sum\limits_{U^-} \alpha_{k-1}(s')\,\gamma_k(s',s)\,\beta_k(s)} \\ &= \log\left[\sum_{U^+} \exp\left(\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s',s) + \tilde{\beta}_k(s)\right)\right] \\ &\quad - \log\left[\sum_{U^-} \exp\left(\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s',s) + \tilde{\beta}_k(s)\right)\right]. \end{aligned} \tag{4.15}$$

It is evident from (4.15) that the constant term in (4.11) may be ignored since it may be factored all the way out of both summations. At first glance, equations (4.10)-(4.15) do not look any

simpler than the probability domain algorithm, but we use the following results to attain the simplification.

Result 4

$$\max(x, y) = \log\left(\frac{e^x + e^y}{1 + e^{-|x-y|}}\right)$$

Proof: Without loss of generality, when $x > y$, the right-hand side equals x. $\square$

Now define

$$\max^*(x, y) \triangleq \log\left(e^x + e^y\right) \tag{4.16}$$

so that from Result 4,

$$\max^*(x, y) = \max(x, y) + \log\left(1 + e^{-|x-y|}\right). \tag{4.17}$$

This may be extended to more than two variables. For example,

$$\max^*(x, y, z) \triangleq \log\left(e^x + e^y + e^z\right)$$

which may be computed pairwise according to the following result.

Result 5

$$\max^*(x, y, z) = \max^*\left[\max^*(x, y), z\right]$$

Proof:

$$\begin{aligned} RHS &= \log\left[e^{\max^*(x,y)} + e^z\right] \\ &= \log\left[e^{\log(e^x+e^y)} + e^z\right] \\ &= \log\left[e^x + e^y + e^z\right] \\ &= LHS \quad \square \end{aligned}$$

Given the function $\max^*(\cdot)$, we may now rewrite (4.10), (4.13), and (4.15) as

$$\tilde{\alpha}_k(s) = \max_{s'}{}^* \left[\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s)\right], \tag{4.18}$$

$$\tilde{\beta}_{k-1}(s') = \max_{s}{}^* \left[\tilde{\beta}_k(s) + \tilde{\gamma}_k(s', s)\right], \tag{4.19}$$

and

$$\begin{aligned} L(u_k) &= \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k(s)\right] \\ &\quad - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k(s)\right]. \end{aligned} \tag{4.20}$$

Fig. 6 illustrates pictorially the trellis-based computations that these last three equations represent.

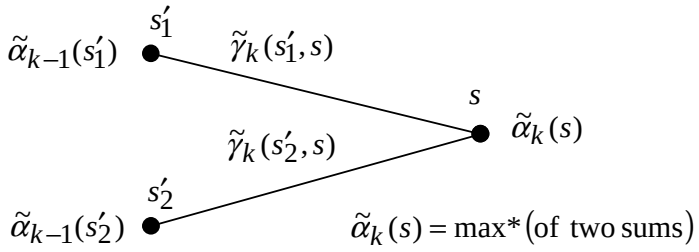
We see from (4.18), (4.19), and (4.20) how the log domain computation of $L(u_k)$ is vastly simplified relative to the probability domain computation. From (4.17), implementation of the $\max^*(\cdot)$ function involves only a two-input $\max(\cdot)$ function plus a lookup table for the "correction term" $\log\left(1 + e^{-|x-y|}\right)$. Robertson *et al.* [11] have shown that a table size of eight is usually sufficient.

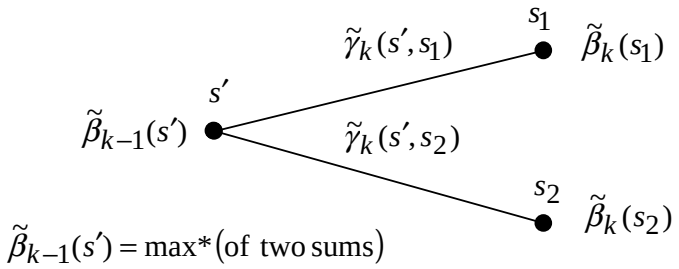
Note that the correction term is bounded as

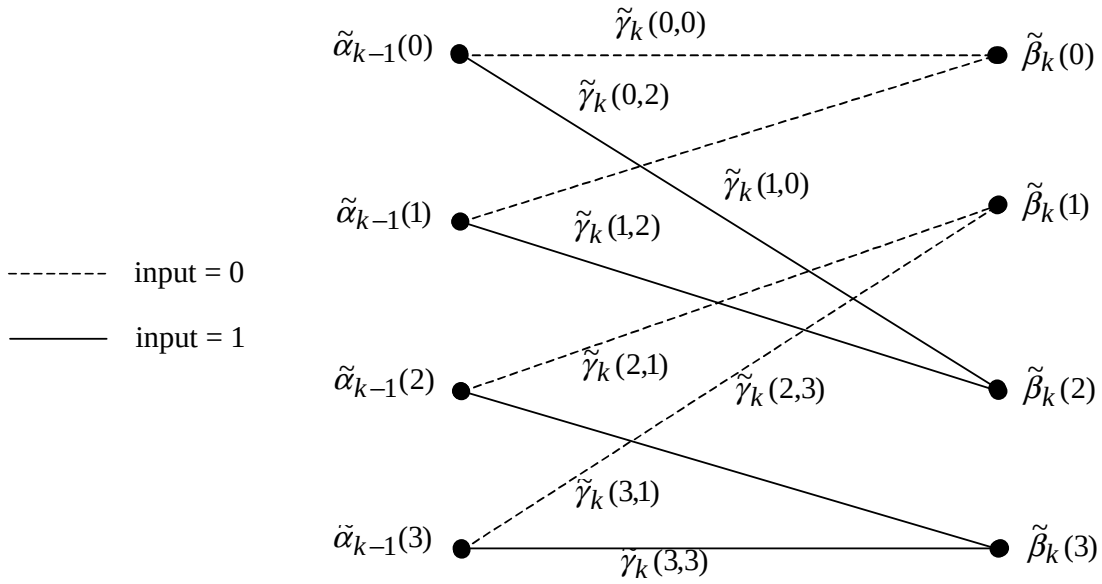
$$0 < \log\left(1 + e^{-|x-y|}\right) \leq \log(2) \simeq 0.693$$

so that $\max^*(x, y) \simeq \max(x, y)$ when $|\max(x, y)| \geq 7$. When $\max^*(x, y)$ is replaced by $\max(\cdot)$ in (4.18) and (4.19), these recursions become forward and reverse Viterbi algorithms, respectively. The performance loss associated with this approximation in turbo decoding depends on the specific turbo code, but a loss of about 0.5 dB is typical [11].

Finally, observe the sense in which the BCJR decoder is a SISO decoder: the decoder input is the "soft-decision" (unquantized) word $\mathbf{y} \in \mathbb{R}^{2K}$ and its outputs are the soft outputs $L(u_k) \in \mathbb{R}$ on which final hard-decisions may be made according to (4.1). Alternatively, in a concatenated code context, these soft outputs may be passed to a companion decoder.







$$L(u_k) = \max^*\{\tilde{\alpha}_{k-1} + \tilde{\gamma}_k + \tilde{\beta}_k \text{ for solid lines}\} - \max^*\{\tilde{\alpha}_{k-1} + \tilde{\gamma}_k + \tilde{\beta}_k \text{ for dashed lines}\}$$

Fig. 6. The top diagram depicts the forward recursion in (4.18), the middle diagram depicts the backward recursion in (4.19), and the bottom diagram depicts the computation of $L(u_k)$ via (4.20).

Summary of the Log-Domain BCJR Algorithm We assume as above a rate 1/2 RSC encoder, a data block $\mathbf{u}$ of length K, and that encoder starts and terminates in the zero state (the last m bits of $\mathbf{u}$ are so selected, where m is the encoder memory size). In practice, the value $-\infty$ used in initialization is simply some large-magnitude negative number.

Initialize $\tilde{\alpha}_0(s)$ and $\tilde{\beta}_K(s)$ according to (4.12) and (4.14).

for $k = 1 : K$

- get $y_k = [y_k^u, y_k^p]$
- compute $\tilde{\gamma}_k(s', s) = -\|y_k - c_k\|^2 / 2\sigma^2$ for all allowable state transitions $s' \to s$ (note $c_k = c_k(s', s)$ here)[2]
- compute $\tilde{\alpha}_k(s)$ for all s using the recursion (4.18)

end

for $k = K : -1 : 2$

- compute $\tilde{\beta}_{k-1}(s')$ for all s' using (4.19)

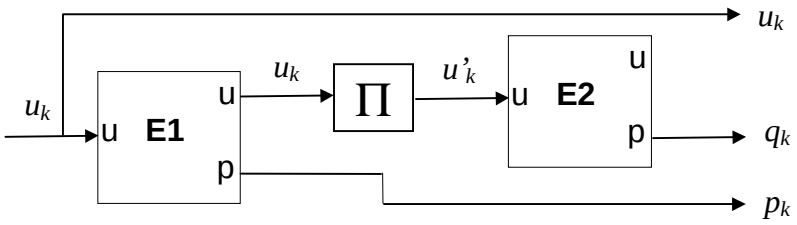
end

for $k = 1 : K$

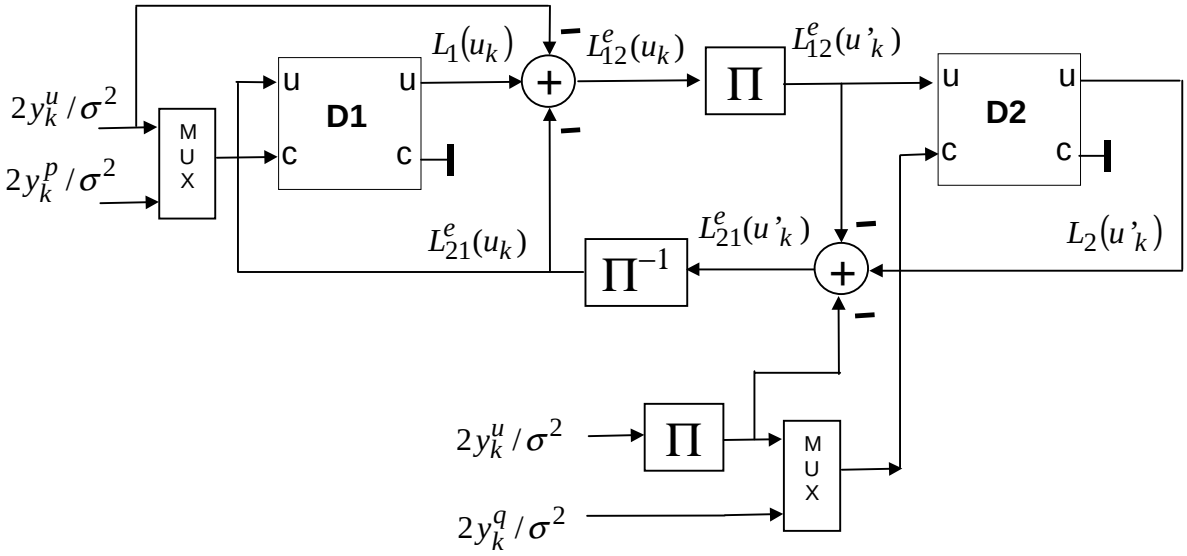
- compute $L(u_k)$ using (4.20)
- compute hard decisions via $\hat{u}_k = \text{sign}\,[L(u_k)]$

end

[2]We may alternatively use $\tilde{\gamma}_k(s', s) = u_k y_k^u / \sigma^2 + p_k y_k^p / \sigma^2$. (See next section.)



(a) PCCC encoder



(b) PCCC iterative decoder

Fig. 7. Block diagrams for the PCCC encoder and iterative decoder.

4.3. The PCCC Iterative Decoder

We present in this section the iterative decoder for a PCCC consisting of two component rate 1/2 RSC encoders concatenated in parallel. We assume no puncturing so that the overall code rate is 1/3. Block diagrams of the PCCC encoder and its iterative decoder with component SISO decoders are presented in Fig. 7. As indicated in Fig. 7(a), the transmitted codeword $\mathbf{c}$ will have the form $\mathbf{c} = [c_1, c_2, ..., c_K] = [u_1, p_1, q_1, ..., u_K, p_K, q_K]$ where $c_k \triangleq [u_k, p_k, q_k]$. The received word $\mathbf{y} = \mathbf{c} + \mathbf{n}$ will have the form $\mathbf{y} = [y_1, y_2, ..., y_K] = [y_1^u, y_1^p, y_1^q, ..., y_K^u, y_K^p, y_K^q]$ where $y_k \triangleq [y_k^u, y_k^p, y_k^q]$, and similarly for $\mathbf{n}$. We denote the codewords produced by E1 and E2 by, respectively, $\mathbf{c}_1 = [c_1^1, c_2^1, ..., c_K^1]$ where $c_k^1 \triangleq [u_k, p_k]$ and $\mathbf{c}_2 = [c_1^2, c_2^2, ..., c_K^2]$ where $c_k^2 \triangleq [u'_k, q_k]$. Note that $\{u'_k\}$ is a permuted version of $\{u_k\}$ and is not actually transmitted (see Fig. 7(a)). We define the noisy received versions of $\mathbf{c}_1$ and $\mathbf{c}_2$ to be $\mathbf{y}_1$ and $\mathbf{y}_2$, respectively, having components $y_k^1 \triangleq [y_k^u, y_k^p]$ and $y_k^2 \triangleq [y_k^{u'}, y_k^q]$, respectively. Note that $\mathbf{y}_1$ and $\mathbf{y}_2$ can be assembled from $\mathbf{y}$ in an obvious fashion (using an interleaver to obtain $\left\{y_k^{u'}\right\}$ from $\{y_k^u\}$). By doing so, the component decoder inputs are the two vectors $\mathbf{y}_1$ and $\mathbf{y}_2$ as indicated in the Fig. 7(b).

In contrast to the BCJR decoder of the previous sections whose input was $\mathbf{y} = \mathbf{c} + \mathbf{n}$ and

whose output was $\{L(u_k)\}$ (or $\{\hat{u}_k\}$), the SISO decoders in Fig. 7(b) possess two inputs and two outputs. The SISO decoders are essentially the BCJR decoders discussed above, except the SISO decoders have the ability to accept from a companion decoder "extrinsic information" about its encoder's input (SISO input label 'u') and/or about its encoder's output (SISO input label 'c'). The SISO decoders also have the ability to produce likelihood information about its encoder's input (SISO output label 'u') and/or about its encoder's output (SISO output label 'c'). Note that the SISO decoder is to be interpreted as a decoding module not all of whose inputs or outputs need be used [7]. (Note the RSC encoders in Fig. 7(a) have also been treated as modules.) As we will see, the SISO modules are connected in a slightly different fashion for the SCCC case.

Note from Fig. 7(b) that the extrinsic information to be passed from D1 to D2 about bit u_k, denoted $L^e_{12}(u_k)$, is equal to the LLR $L_1(u_k)$ produced by D1 minus the channel likelihood $2y^u_k/\sigma^2$ and the extrinsic information $L^e_{21}(u_k)$ that D1 had just received from D2. The idea is that $L^e_{12}(u_k)$ should indeed be extrinsic (and uncorrelated with) the probabilistic information already possessed by D2. As we will see, $L^e_{12}(u_k)$ is strictly a function of received E1 parity $\{y^p_k\}$ which is not directly sent to D2. Observe that $\{L^e_{12}(u_k)\}$ must be interleaved prior to being sent to D2 since E2 and D2 operate on the interleaved data bits u'_k. Symmetrical comments may be made about the extrinsic information to be passed from D2 to D1, $L^e_{21}(u_k)$ (e.g., it is a function of E2 parity and de-interleaving is necessary).

We already know from the previous section how the SISO decoders process the samples from the channel, $\mathbf{y}_i$ $(i = 1, 2)$, to obtain LLR's about the decoder inputs. We need now to discuss how the SISO decoders include the extrinsic information in their computations. As indicated earlier, the extrinsic information takes the role of *a priori* information in the iterative decoding algorithm (cf. (4.2) and surrounding discussion),

$$L^e(u_k) \triangleq \log\left(\frac{P(u_k = +1)}{P(u_k = -1)}\right). \tag{4.21}$$

The *a priori* term $P(u_k)$ shows up in (4.8) in an expression for $\gamma_k(s', s)$. In the log domain, (4.8) becomes[3]

$$\tilde{\gamma}_k(s', s) = \log P(u_k) - \log\left(\sqrt{2\pi}\sigma\right) - \frac{\|y_k - c_k\|^2}{2\sigma^2}. \tag{4.22}$$

Now observe that we may write

$$\begin{aligned} P(u_k) &= \left(\frac{\exp[-L^e(u_k)/2]}{1 + \exp[-L^e(u_k)]}\right) \cdot \exp[u_k L^e(u_k)/2] \\ &= A_k \exp[u_k L^e(u_k)/2] \end{aligned} \tag{4.23}$$

where the first equality follows since it equals

$$\begin{aligned} \left(\frac{\sqrt{P_-/P_+}}{1 + P_-/P_+}\right)\sqrt{P_+/P_-} &= P_+ \text{ when } u_k = +1 \text{ and} \\ \left(\frac{\sqrt{P_-/P_+}}{1 + P_-/P_+}\right)\sqrt{P_-/P_+} &= P_- \text{ when } u_k = -1, \end{aligned}$$

[3]For the time being, we will discuss a generic SISO decoder so that we may avoid using cumbersome superscripts until it is necessary to do so.

where we have defined $P_+ \triangleq P(u_k = +1)$ and $P_- \triangleq P(u_k = -1)$ for convenience. Substitution of (4.23) into (4.22) yields

$$\tilde{\gamma}_k(s', s) = \log\left(A_k/\sqrt{2\pi}\sigma\right) + u_k L^e(u_k)/2 - \frac{\|y_k - c_k\|^2}{2\sigma^2} \tag{4.24}$$

where we will see that the first term may be ignored.

Thus, the extrinsic information received from a companion decoder is included in the computation through the branch metric $\tilde{\gamma}_k(s', s)$. The rest of the BCJR/SISO algorithm proceeds as before using equations (4.18), (4.19), and (4.20).

Upon substitution of (4.24) into (4.20), we have

$$\begin{aligned} L(u_k) &= L^e(u_k) + \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}(s') + u_k y_k^u/\sigma^2 + p_k y_k^p/\sigma^2 + \tilde{\beta}_k(s)\right] \\ &\quad - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}(s') + u_k y_k^u/\sigma^2 + p_k y_k^p/\sigma^2 + \tilde{\beta}_k(s)\right] \end{aligned} \tag{4.25}$$

where we have use the fact that

$$\begin{aligned} \|y_k - c_k\|^2 &= (y_k^u - u_k)^2 + (y_k^p - p_k)^2 \\ &= (y_k^u)^2 - 2u_k y_k^u + u_k^2 + (y_k^p)^2 - 2p_k y_k^p + p_k^2 \end{aligned}$$

and only the terms dependent on U^+ or U^-, $u_k y_k^u/\sigma^2$ and $p_k y_k^p/\sigma^2$, survive after the subtraction. Now note that $u_k y_k^u/\sigma^2 = y_k^u/\sigma^2$ under the first $\max^*(\cdot)$ operation in (4.25) (U^+ is the set of state transitions for which $u_k = +1$) and $u_k y_k^u/\sigma^2 = -y_k^u/\sigma^2$ under the second $\max^*(\cdot)$ operation. Using the definition for $\max^*(\cdot)$, it is easy to see that these terms may be isolated out so that

$$\begin{aligned} L(u_k) &= 2y_k^u/\sigma^2 + L^e(u_k) + \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}(s') + p_k y_k^p/\sigma^2 + \tilde{\beta}_k(s)\right] \\ &\quad - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}(s') + p_k y_k^p/\sigma^2 + \tilde{\beta}_k(s)\right]. \end{aligned} \tag{4.26}$$

The interpretation of this new expression for $L(u_k)$ is that the first term is likelihood information received directly from the channel, the second term is extrinsic likelihood information received from a companion decoder, and the third "term" $(\max_{U^+}{}^* - \max_{U^-}{}^*)$ is extrinsic likelihood information to be passed to a companion decoder. Note that this third term is likelihood information gleaned from received parity not available to the companion decoder. Thus, specializing to decoder D1, for example, on any given iteration, D1 computes

$$L_1(u_k) = 2y_k^u/\sigma^2 + L_{21}^e(u_k) + L_{12}^e(u_k)$$

where $L_{21}^e(u_k)$ is extrinsic information received from D2, and $L_{12}^e(u_k)$ is the third term in (4.26) which is to be used as extrinsic information from D1 to D2.

4.3.1. Summary of the PCCC Iterative Decoder

The algorithm given below for the iterative decoding of a parallel turbo code follows directly from the development above. The constituent decoder order is D1, D2, D1, D2, etc. Implicit is the fact that each decoder must have full knowledge of the trellis of the constituent encoders. For example, each decoder must have a table (array) containing the input bits and parity bits for all possible state transitions $s' \rightarrow s$. Also required are interleaver and de-interleaver functions (arrays) since D1 and D2 will be sharing reliability information about each u_k, but D2's information is permuted relative to D1. We denote these arrays by $P[\cdot]$ and $Pinv[\cdot]$, respectively. For example, the permuted word $\mathbf{u}'$ is obtained from the original word $\mathbf{u}$ via the pseudo-code statement: for $k = 1 : K$, $u'_k = u_{P[k]}$, end.

We next point out that knowledge of the noise variance $\sigma^2 = N_0/2$ by each SISO decoder is necessary. Also, a simple way to obtain higher code rates via (simulated) puncturing is, in the computation of $\gamma_k(s', s)$, to set to zero the received parity samples, y_k^p or y_k^q, corresponding to the punctured parity bits, p_k or q_k. (This will set to zero the term in the branch metric corresponding to the punctured bit.) Thus, puncturing need not be performed at the encoder for computer simulations. We mention also that termination of encoder E2 to the zero state can be problematic due to the presence of the interleaver (for one solution, see [19]). Fortunately, there is only a small performance loss when E2 is not terminated. In this case, $\beta_K(s)$ for D2 may be set to $\alpha_K(s)$ for all s, or it may be set to a nonzero constant (e.g., $1/S_2$, where S_2 is the number of E2 states).

Finally, we remark that some sort of iteration stopping criterion is necessary. The most straightforward criterion is to set a maximum number of iterations. However, this can be inefficient since the correct codeword is often found after only two or three iterations. Another straightforward technique is to utilize a carefully chosen outer error detection code. After each iteration, a parity check is made and the iterations stop whenever no error is detected. Other stopping criteria are presented in [9] and elsewhere in the literature.

Initialization
D1:

$$\tilde{\alpha}_0^{(1)}(s) = 0 \text{ for } s = 0$$
$$= -\infty \text{ for } s \neq 0$$

$$\tilde{\beta}_K^{(1)}(s) = 0 \text{ for } s = 0$$
$$= -\infty \text{ for } s \neq 0$$

$$L_{21}^e(u_k) = 0 \text{ for } k = 1, 2, ..., K$$

D2:

$$\tilde{\alpha}_0^{(2)}(s) = 0 \text{ for } s = 0$$
$$= -\infty \text{ for } s \neq 0$$

$\tilde{\beta}_K^{(2)}(s) = \tilde{\alpha}_K^{(2)}(s)$ for all s (set once after computation of $\{\tilde{\alpha}_K^{(2)}(s)\}$ in the *first* iteration)

$L_{12}^e(u_k)$ is to be determined from D1 after the first half-iteration and so need not be initialized

The n^{th} Iteration
D1:

for $k = 1 : K$

- get $y_k^1 = [y_k^u, y_k^p]$
- compute $\tilde{\gamma}_k(s', s)$ for all allowable state transitions $s' \to s$ from (4.24) which simplifies to (see discussion following (4.24))

$$\tilde{\gamma}_k(s', s) = u_k L_{21}^e(u_{Pinv[k]})/2 + u_k y_k^u/\sigma^2 + p_k y_k^p/\sigma^2$$

  [u_k (p_k) in this expression is set to the value of the encoder input (output) corresponding to the transition $s' \to s$]
- compute $\tilde{\alpha}_k^{(1)}(s)$ for all s using (4.18)

end

for $k = K : -1 : 2$

- compute $\tilde{\beta}_{k-1}^{(1)}(s)$ for all s using (4.19)

end

for $k = 1 : K$

- compute $L_{12}^e(u_k)$ using[4]

$$L_{12}^e(u_k) = \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + p_k y_k^p/\sigma^2 + \tilde{\beta}_k^{(1)}(s)\right] - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + p_k y_k^p/\sigma^2 + \tilde{\beta}_k^{(1)}(s)\right]$$

end

D2:

for $k = 1 : K$

- get $y_k^2 = [y_{P[k]}^u, y_k^q]$

[4]Note here we are computing $L_{12}^e(u_k)$ directly rather than computing $L_1(u_k)$ and then subtracting $2y_k^u/\sigma^2 + L_{21}^e(u_k)$ from it to obtain $L_{12}^e(u_k)$ as in Fig. 5(b). We will do likewise in the analogous step for D2.

– compute $\tilde{\gamma}_k(s', s)$ for all allowable state transitions $s' \rightarrow s$ from

$$\tilde{\gamma}_k(s', s) = u_k L_{12}^e(u_{P[k]})/2 + u_k y_{P[k]}^u/\sigma^2 + q_k y_k^q/\sigma^2$$

[u_k (q_k) in this expression is set to the value of the encoder input (output) corresponding to the transition $s' \rightarrow s$]

– compute $\tilde{\alpha}_k^{(2)}(s)$ for all s using (4.18)

end

for $k = K : -1 : 2$

– compute $\tilde{\beta}_{k-1}^{(2)}(s)$ for all s using (4.19)

end

for $k = 1 : K$

– compute $L_{21}^e(u_k)$ using

$$L_{21}^e(u_k) = \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + q_k y_k^q/\sigma^2 + \tilde{\beta}_k^{(2)}(s)\right] - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + q_k y_k^q/\sigma^2 + \tilde{\beta}_k^{(2)}(s)\right]$$

end

After the Last Iteration

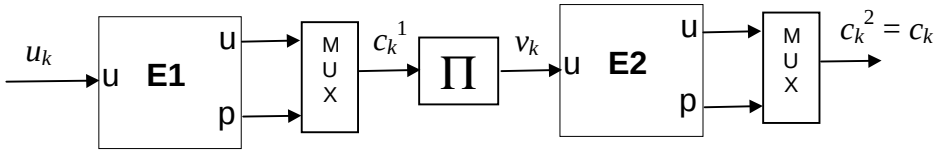
for $k = 1 : K$

– compute

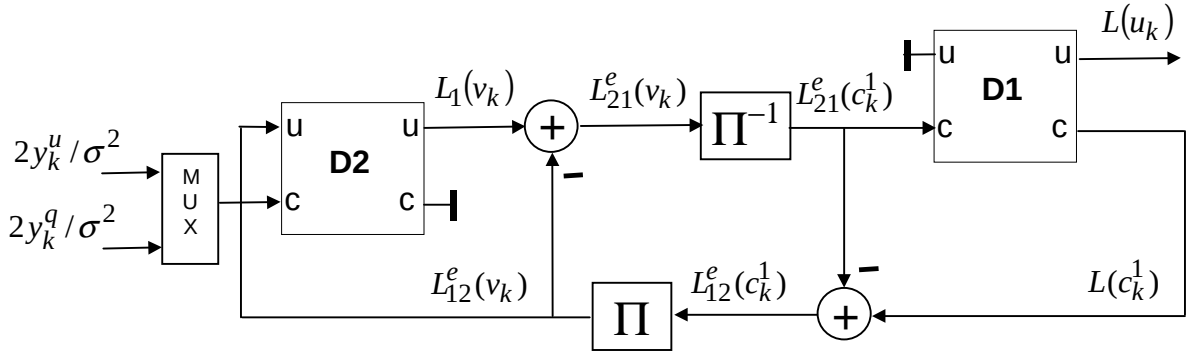
$$L_1(u_k) = 2y_k^u/\sigma^2 + L_{21}^e(u_{Pinv[k]}) + L_{12}^e(u_k)$$

– $\hat{u}_k = \text{sign}\,[L(u_k)]$

end



(a) SCCC encoder



(b) SCCC iterative decoder

Fig. 8. Block diagrams for the SCCC encoder and iterative decoder.

4.4. The SCCC Iterative Decoder

We present in this section the iterative decoder for an SCCC consisting of two component rate 1/2 RSC encoders concatenated in series. We assume no puncturing so that the overall code rate is 1/4. Higher code rates are achievable via puncturing and/or by replacing the inner encoder with a rate 1 differential encoder with transfer function $\frac{1}{1\oplus D}$. It is straightforward to derive the iterative decoding algorithm for other SCCC codes from the special case that we consider here.

Block diagrams of the SCCC encoder and its iterative decoder with component SISO decoders are presented in Fig. 8. We denote by $\mathbf{c}_1 = [c_1^1, c_2^1, ..., c_{2K}^1] = [u_1, p_1, u_2, p_2, ..., u_K, p_K]$ the codeword produced by E1 whose input is $\mathbf{u} = [u_1, u_2, ..., u_K]$. We denote by $\mathbf{c}_2 = [c_1^2, c_2^2, ..., c_{2K}^2] = [v_1, q_1, v_2, q_2, ..., v_{2K}, q_{2K}]$ (with $c_k^2 \triangleq [v_k, q_k]$) the codeword produced by E2 whose input $\mathbf{v} = [v_1, v_2, ..., v_{2K}]$ is the interleaved version of $\mathbf{c}_1$, that is, $\mathbf{v} = \mathbf{c}_1'$. As indicated in Fig.8(a), the transmitted codeword $\mathbf{c}$ is the codeword $\mathbf{c}_2$. The received word $\mathbf{y} = \mathbf{c} + \mathbf{n}$ will have the form $\mathbf{y} = [y_1, y_2, ..., y_{2K}] = [y_1^v, y_1^q, ..., y_{2K}^v, y_{2K}^q]$ where $y_k \triangleq [y_k^v, y_k^q]$, and similarly for $\mathbf{n}$.

The iterative SCCC decoder in Fig. 8(b) employs two SISO decoding modules (described in the previous section). Note that unlike the PCCC case, these SISO decoders share extrinsic information on the code bits $\{c_k^1\}$ (equivalently, on the input bits $\{v_k\}$) in accordance with the fact that these are the bits known to both encoders. A consequence of this is that D1 must provide likelihood information on E1 *output* bits whereas D2 produces likelihood information on E2 *input* bits as indicated in Fig. 8(b). However, because LLRs must be obtained on the original data bits u_k so that final decisions may be made, D1 must also compute likelihood information on E1 input bits. Note also that, because E1 feeds no bits directly to the channel,

D1 receives no samples directly from the channel. Instead, the only input to D1 is the extrinsic information it receives from D2.

Thus, the SISO module D1 requires two features that we have not discussed in any detail to this point. The first is providing likelihood information on the encoder's input *and* output. However, since we assume the component codes are systematic, we need only compute LLRs on the encoder's output bits $[u_1, p_1, u_2, p_2, ..., u_K, p_K]$. Doing this is a simple matter of modifying the summation indices in (4.3) to those relevant to the output bit of interest. For example, the LLR $L(p_k)$ for the E1 parity bit p_k is obtained via

$$L(p_k) = \log \frac{\sum\limits_{P^+} p\left(s_{k-1} = s', s_k = s, \mathbf{y}\right)}{\sum\limits_{P^-} p\left(s_{k-1} = s', s_k = s, \mathbf{y}\right)} \tag{4.27}$$

where P^+ is set of state transition pairs (s', s) corresponding to the event $p_k = +1$, and P^- is similarly defined. (A trellis-based BCJR/SISO decoder is generally capable of decoding either the encoder's input or its output, whether or not the code is systematic. This is evident since the trellis branches are labeled by both inputs and outputs.)

The second feature is required by D1 is decoding with only extrinsic information as input. In this case the branch metric is simply modified as (cf. (4.24))

$$\tilde{\gamma}_k(s', s) = u_k L_{21}^e(u_k)/2 + p_k L_{21}^e(p_k)/2. \tag{4.28}$$

Other than these modifications, the iterative SCCC decoder proceeds much like the PCCC iterative decoder and as indicated in Fig. 8(b).

4.4.1. Summary of the SCCC Iterative Decoder

Essentially all of the comments mentioned for the PCCC decoder hold here as well and so we do not repeat them. The only difference is that the decoding order is D2, D1, D2, D1, etc.

Initialization
D1

$\tilde{\alpha}_0^{(1)}(s) = 0$ for $s = 0$
$\quad = -\infty$ for $s \neq 0$

$\tilde{\beta}_K^{(1)}(s) = 0$ for $s = 0$
$\quad = -\infty$ for $s \neq 0$

$L_{21}^e(c_k^1)$ is to be determined from D2 after the first half-iteration and so need not be initialized

D2:

$\tilde{\alpha}_0^{(2)}(s) = 0$ for $s = 0$
$\quad = -\infty$ for $s \neq 0$

$\tilde{\beta}_{2K}^{(2)}(s) = \tilde{\alpha}_{2K}^{(2)}(s)$ for all s (set after computation of $\{\tilde{\alpha}_{2K}^{(2)}(s)\}$ in the *first* iteration)

$L_{12}^e(v_k) = 0$ for $k = 1, 2, ..., 2K$

The n^{th} Iteration
D2:

for $k = 1 : 2K$

- get $y_k = [y_k^v, y_k^q]$
- compute $\tilde{\gamma}_k(s', s)$ for all allowable state transitions $s' \to s$ from

$$\tilde{\gamma}_k(s', s) = v_k L_{12}^e(v_k)/2 + v_k y_k^v/\sigma^2 + q_k y_k^q/\sigma^2$$

  [v_k (q_k) in the above expression is set to the value of the encoder input (output) corresponding to the transition $s' \to s$; $L_{12}^e(v_k)$ is $L_{12}^e(c_{P[k]}^1)$, the interleaved extrinsic information from the previous D1 iteration.]

- compute $\tilde{\alpha}_k^{(2)}(s)$ for all s using (4.18)

end

for $k = 2K : -1 : 2$

- compute $\tilde{\beta}_{k-1}^{(2)}(s)$ for all s using (4.19)

end

for $k = 1 : 2K$

- compute $L_{21}^e(v_k)$ using

$$\begin{aligned} L_{21}^e(v_k) &= \max_{V^+}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k^{(2)}(s)\right] \\ &\quad - \max_{V^-}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k^{(2)}(s)\right] - L_{12}^e(v_k) \\ &= \max_{V^+}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + v_k y_k^v/\sigma^2 + q_k y_k^q/\sigma^2 + \tilde{\beta}_k^{(2)}(s)\right] \\ &\quad - \max_{V^-}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + v_k y_k^v/\sigma^2 + q_k y_k^q/\sigma^2 + \tilde{\beta}_k^{(2)}(s)\right] \end{aligned}$$

  where V^+ is set of state transition pairs (s', s) corresponding to the event $v_k = +1$, and V^- is similarly defined.

end

D1:

for $k = 1 : K$

– for all allowable state transitions $s' \to s$ set $\tilde{\gamma}_k(s', s)$ via

$$\begin{aligned}\tilde{\gamma}_k(s', s) &= u_k L_{21}^e(u_k)/2 + p_k L_{21}^e(p_k)/2 \\ &= u_k L_{21}^e(c_{2k-1}^1)/2 + p_k L_{21}^e(c_{2k}^1)/2\end{aligned}$$

[u_k (p_k) in the above expression is set to the value of the encoder input (output) corresponding to the transition $s' \to s$; $L_{21}^e(c_{2k-1}^1)$ is $L_{21}^e(v_{Pinv[2k-1]})$, the de-interleaved extrinsic information from the previous D2 iteration, and similarly for $L_{21}^e(c_{2k}^1)$.]

– compute $\tilde{\alpha}_k^{(1)}(s)$ for all s using (4.18)

end

for $k = K : -1 : 2$

– compute $\tilde{\beta}_{k-1}^{(1)}(s)$ for all s using (4.19)

end

for $k = 1 : K$

– compute $L_{12}^e(u_k) = L_{12}^e(c_{2k-1}^1)$ using

$$\begin{aligned}L_{12}^e(u_k) = &\max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k^{(1)}(s)\right] \\ - &\max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k^{(1)}(s)\right] - L_{21}^e(c_{2k-1}^1) \\ = &\max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + p_k L_{21}^e(p_k)/2 + \tilde{\beta}_k^{(1)}(s)\right] \\ - &\max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + p_k L_{21}^e(p_k)/2 + \tilde{\beta}_k^{(1)}(s)\right]\end{aligned}$$

– compute $L_{12}^e(p_k) = L_{12}^e(c_{2k}^1)$ using

$$\begin{aligned}L_{12}^e(p_k) = &\max_{P^+}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k^{(1)}(s)\right] \\ - &\max_{P^-}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k^{(1)}(s)\right] - L_{21}^e(c_{2k}^1) \\ = &\max_{P^+}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + u_k L_{21}^e(u_k)/2 + \tilde{\beta}_k^{(1)}(s)\right] \\ - &\max_{P^-}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + u_k L_{21}^e(u_k)/2 + \tilde{\beta}_k^{(1)}(s)\right]\end{aligned}$$

end

After the Last Iteration

for $k = 1 : K$

– for all allowable state transitions $s' \to s$ set $\tilde{\gamma}_k(s', s)$ via

$$\tilde{\gamma}_k(s', s) = u_k L_{21}^e(c_{2k-1}^1)/2 + p_k L_{21}^e(c_{2k}^1)/2$$

– compute $L(u_k)$ using

$$L(u_k) = \max_{U^+}{}^* \left[\tilde{\alpha}^{(1)}_{k-1}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}^{(1)}_k(s)\right] - \max_{U^-}{}^* \left[\tilde{\alpha}^{(1)}_{k-1}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}^{(1)}_k(s)\right]$$

– $\hat{u}_k = \text{sign}\,[L(u_k)]$

end

5. Conclusion

We have seen in this chapter the how and why of both parallel and serial turbo codes. That is, we have seen how to decode these codes using an iterative decoder, and why they should be expected to perform so well. The decoding algorithm summaries should be sufficient to decode any binary parallel and serial turbo codes, and can in fact be easily extended to the iterative decoding of any binary hybrid schemes. It is not much more work to figure out how to decode any of the turbo trellis-coded modulation (turbo-TCM) schemes that appear in the literature. In any case, this chapter should serve well as a starting point for the study of concatenated codes (and perhaps graph-based codes) and their iterative decoders.

Acknowledgment

The author would like to thank Rajeev Ramamurthy and Bo Xia for producing Figs. 4 and 5, and Steve Wilson, Masoud Salehi, and John Proakis for helpful comments. He would also like to thank Cheryl Drier for typing the first draft.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error- correcting coding and decoding: Turbo codes," *Proc. 1993 Int. Conf. Comm.*, pp. 1064- 1070.

[2] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Comm.*, pp. 1261-1271, Oct. 1996.

[3] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inf. Theory*, pp. 55-67, Jan. 1982.

[4] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inf. Theory*, pp. 409-428, Mar. 1996.

[5] S. Benedetto and G. Montorsi, "Design of parallel concatenated codes," *IEEE Trans. Comm.*, pp. 591-600, May 1996.

[6] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *IEEE Trans. Inf. Theory*, pp. 909-926, May 1998.

[7] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," TDA Progress Report 42-127, Nov. 15, 1996.

[8] D. Divsalar and F. Pollara, "Multiple turbo codes for deep-space communications," JPL TDA Progress Report, 42-121, May 15, 1995.

[9] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, pp. 429-445, Mar. 1996.

[10] L. Perez, J. Seghers, and D. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inf. Theory*, pp. 1698-1709, Nov. 1996.

[11] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and suboptimal MAP decoding algorithms operating in the log domain," *Proc. 1995 Int. Conf. on Comm.*, pp. 1009-1013.

[12] A. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE JSAC*, pp. 260-264, Feb. 1998.

[13] O. Acikel and W. Ryan, "Punctured turbo codes for BPSK/QPSK channels," *IEEE Trans. Comm.*, pp. 1315-1323, Sept. 1999.

[14] O. Acikel and W. Ryan, "Punctured high rate SCCCs for BPSK/QPSK channels," *Proc. 2000 IEEE International Conference on Communications*, vol. 1, pp. 434-439.

[15] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, San Mateo, CA: Morgan Kaufmann, 1988.

[16] B. Frey, *Graphical Models for Machine Learning and Digital Communication*, MIT Press, 1998.

[17] N. Wiberg, *Codes and Decoding on General Graphs*, Ph.D. dissertation, U. Linköping, Sweden, 1996.

[18] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, pp. 284-287, Mar. 1974.

[19] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," *Proc. 1995 Int. Conf. Comm.*, pp. 54-59.

Transactions Papers

Near Optimum Error Correcting Coding And Decoding: Turbo-Codes

Claude Berrou, *Member, IEEE*, and Alain Glavieux

***Abstract*—This paper presents a new family of convolutional codes, nicknamed turbo-codes, built from a particular concatenation of two recursive systematic codes, linked together by nonuniform interleaving. Decoding calls on iterative processing in which each component decoder takes advantage of the work of the other at the previous step, with the aid of the original concept of extrinsic information. For sufficiently large interleaving sizes, the correcting performance of turbo-codes, investigated by simulation, appears to be close to the theoretical limit predicted by Shannon.**

I. Introduction

CONVOLUTIONAL error correcting or channel coding has become widespread in the design of digital transmission systems. One major reason for this is the possibility of achieving real-time decoding without noticeable information losses thanks to the well-known soft-input Viterbi algorithm [1]. Moreover, the same decoder may serve for various coding rates by means of puncturing [2], allowing the same silicon product to be used in different applications. Two kinds of convolutional codes are of practical interest: nonsystematic convolutional (NSC) and recursive systematic convolutional (RSC) codes. Though RSC codes have the same free distance d_f as NSC codes and exhibit better performance at low signal to noise ratios (SNR's) and/or when punctured, only NSC codes have actually been considered for channel coding, except in Trellis-coded modulation (TCM) [3]. Section II presents the principle and the performance of RSC codes, which are at the root of the study expounded in this article.

For a given rate, the error-correcting power of convolutional codes, measured as the coding gain at a certain binary error rate (BER) in comparison with the uncoded transmission, grows more or less linearly with code memory ν. Fig. 1 (from [4]) shows the achievable coding gains for different rates, and corresponding bandwidth expansion rates, by using classical NSC codes with $\nu = 2, 4, 6$ and 8, for a BER of 10^{-6}. For instance, with $R = 1/2$, each unit added to ν adds about 0.5 dB more to the coding gain, up to $\nu = 6$; for $\nu = 8$, the additional gain is lower. Unfortunately, the complexity of the decoder is not a linear function of ν and it grows exponentially as $\nu \cdot 2^{\nu}$. Factor 2 represents the number of states processed by the decoder and the multiplying factor ν accounts for the complexity of the memory part (metrics and survivor memory). Other technical limitations like the interconnection constraint in the silicon decoder lay-out, make the value of six a practical upper limit for ν for most applications, especially for high data rates.

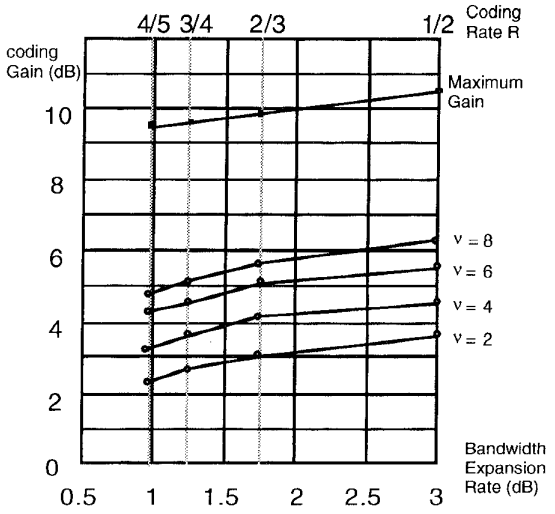


Fig. 1. Coding gains at BER equal to 10^{-6}, achievable with NSC codes (three-bit quantization, from [4], and maximum possible gains for 1/2, 2/3, 3/4, and 4/5 rates in a Gaussian channel, with quaternary phase shift keying (QPSK) modulation.

In order to obtain high coding gains with moderate decoding complexity, concatenation has proved to be an attractive scheme. Classically, concatenation has consisted in cascading a block code (the outer code, typically a Reed-Solomon code) and a convolutional code (the inner code) in a serial structure. Another concatenated code, which has been given the familiar name of *turbo-code,* with an original parallel organization of two RSC elementary codes, is described in Section III. Some comments about the distance properties of this composite

Paper approved by D. Divsalar, the Editor for Coding Theory and Applications of the IEEE Communications Society. Manuscript received March 3, 1995; revised December 6, 1995 and February 26, 1996. This paper was presented at ICC'93, Geneva, Switzerland, May 1993.

The authors are with Ecole Nationale Supérieure des Télécommunications de Bretagne, BP 832 29285 BREST CEDEX, France.

Publisher Item Identifier S 0090-6778(96)07373-4.

code, are propounded. When decoded by an iterative process, turbo-codes offer near optimum performance. The way to achieve this decoding with the *Maximum A Posteriori* (MAP) algorithm is detailed in Sections IV, V, VI, and some basic results are given in Section VII.

II. Recursive Systematic Convolutional Codes

A. Introduction

Consider a binary rate $R = 1/2$ convolutional encoder with constraint length K and memory $\nu = K - 1$. The input to the encoder at time k is a bit d_k and the corresponding binary couple (X_k, Y_k) is equal to

$$X_k = \sum_{i=0}^{\nu} g_{1i} d_{k-i} \qquad g_{1i} = 0, 1 \tag{1a}$$

$$Y_k = \sum_{i=0}^{\nu} g_{2i} d_{k-i} \qquad g_{2i} = 0, 1 \tag{1b}$$

where $G_1{:}\{g_{1i}\}, G_2{:}\{g_{2i}\}$ are the two encoder generators, expressed in octal form. It is well known that the BER of a classical NSC code is lower than that of a classical nonrecursive systematic convolutional code with the same memory ν at large SNR's, since its free distance is smaller than that of a NSC code [5]. At low SNR's, it is in general the other way round. The RSC code, presented below, combines the properties of NSC and systematic codes. In particular, it can be better than the equivalent NSC code, at any SNR, for code rates larger than 2/3.

A binary rate RSC code is obtained from a NSC code by using a feedback loop and setting one of the two outputs X_k or Y_k equal to the input bit d_k. The shift register (memory) input is no longer the bit d_k but is a new binary variable a_k. If $X_k = d_k$ (respectively, $Y_k = d_k$), the output Y_k (resp. X_k) is defined by (1b) [respectively, (1a)] by substituting d_k for a_k and variable a_k is recursively calculated as

$$a_k = d_k + \sum_{i=1}^{\nu} \gamma_i a_{k-i} \tag{2}$$

where γ_i is respectively equal to g_{1i} if $X_k = d_k$ and to g_{2i} if $Y_k = d_k$. Equation (2) can be rewritten as

$$d_k = \sum_{i=0}^{\nu} \gamma_i a_{k-i}. \tag{3}$$

Taking into account $X_k = d_k$ or $Y_k = d_k$, the RSC encoder output $C_k = (X_k, Y_k)$ has exactly the same expression as the NSC encoder outputs if $g_{10} = g_{20} = 1$ and by substituting d_k for a_k in (1a) or (1b).

Two RSC encoders with memory $\nu = 2$ and rate $R = 1/2$, obtained from a NSC encoder defined by generators $G_1 = 7, G_2 = 5$, are depicted in Fig. 2.

Generally, we assume that the input bit d_k takes values zero or one with the same probability. From (2), we can show that variable a_k exhibits the same statistical property

$$\begin{aligned} &\Pr\{a_k = 0/a_{k-\nu} = \varepsilon_\nu, \cdots a_{k-i} = \varepsilon_i, \cdots a_{k-1} = \varepsilon_1\} \\ &\quad = \Pr\{d_k = \varepsilon\} = \tfrac{1}{2} \end{aligned} \tag{4}$$

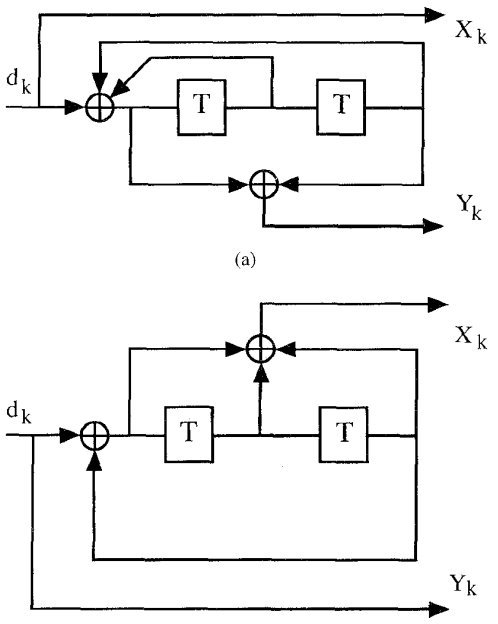


Fig. 2. Two associated Recursive systematic convolutional (RSC) encoders with memory $\nu = 2$, rate $R = 1/2$ and generators $G_1 = 7, G_2 = 5$.

where ε is equal to

$$\varepsilon = \sum_{I=1}^{\nu} \gamma_i \varepsilon_i; \quad \varepsilon_i = 0, 1. \tag{5}$$

Thus, the transition state probabilities $\pi(\boldsymbol{S}_k = \boldsymbol{m}/\boldsymbol{S}_{k-1} = \boldsymbol{m}')$, where $\boldsymbol{S}_k = \boldsymbol{m}$ and $\boldsymbol{S}_{k-1} = \boldsymbol{m}'$ are, respectively, the encoder state at time k and at time $(k-1)$, are identical for the equivalent RSC and NSC codes; moreover these two codes have the same free distance d_f. However, for a same input sequence $\{d_k\}$, the two output sequences $\{X_k\}$ and $\{Y_k\}$ are different for RSC and NSC codes.

When puncturing is considered, some output bits X_k or Y_k are deleted according to a chosen perforation pattern defined by a matrix $\boldsymbol{P}$. For instance, starting from a rate $R = 1/2$ code, the matrix $\boldsymbol{P}$ of rate 2/3 punctured code can be equal to

$$\boldsymbol{P} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}. \tag{6}$$

For the punctured RSC code, bit d_k must be emitted at each time k. This is obviously done if all the elements belonging to the first or second row of matrix $\boldsymbol{P}$ are equal to one. When the best perforation pattern for a punctured NSC code is such that matrix $\boldsymbol{P}$ has null elements in the first and the second rows, the punctured recursive convolutional code is no longer systematic. In order to use the same matrix $\boldsymbol{P}$ for both RSC and NSC codes, the RSC encoder is now defined by (3), (7), and (8)

$$Y_k = \sum_{i=0}^{\nu} \lambda_i a_{k-i} \tag{7}$$

$$X_k = d_k \tag{8}$$

where coefficients γ_i [see (3)] and λ_i are, respectively, equal to g_{1i} and g_{2i} when element $p_{1j}; 1 \leq j \leq n$ of matrix $\boldsymbol{P}$ is equal to one and to g_{2i} and g_{1i} when p_{1j} is equal to zero.

B. Recursive Systematic Code Performance

In order to compare the performance of RSC and NSC codes, we determined their weight spectrum and their BER. The weight spectrum of a code is made up of two sets of coefficients $a(d)$ and $W(d)$ obtained from two series expansions related to the code transfer function $T(D, N)$ [6]

$$T(D,N)\Big|_{N=1} = \sum_{d=d_f}^{\infty} a(d)D^d \tag{9}$$

$$\frac{\partial T(D,N)}{\partial N}\Big|_{N=1} = \sum_{d=d_f}^{\infty} W(d)D^d \tag{10}$$

where d_f is the free distance of the code, $a(d)$ is the number of paths at Hamming distance d from the "null" path and $W(d)$ is the total Hamming weight of input sequences $\{d_k\}$ used to generate all paths at distance d from the "null" path. In general, it is not easy to calculate the transfer function of a punctured code, that is why the first coefficients $a(d)$ and $W(d)$ are directly obtained from the trellis by using an algorithm derived from [7]. From coefficients $W(d)$, a tight upper bound of error probability can be calculated for large SNR's [6]

$$P_e \leq \sum_{d=d_f}^{\infty} W(d)P(d). \tag{11}$$

For a memoryless Gaussian channel with binary modulation (PSK, QPSK), probability $P(d)$ is equal to

$$P(d) = \frac{1}{2}\operatorname{erfc}\left[\sqrt{dR\frac{E_b}{N_0}}\right] \tag{12}$$

where E_b/N_0 is the energy per information bit to noise power spectral density ratio and R is the code rate.

In [8], a large number of RSC codes have been investigated and their performance was compared to that of NSC codes, in term of weight spectrum and of BER. Coefficients $a(d)$ are the same for RSC and NSC codes but the coefficients $\{W_{\mathrm{RSC}}(d)\}$ of RSC codes have a tendency to increase more slowly in function of d than the coefficients $\{W_{\mathrm{NSC}}(d)\}$ of NSC codes, whatever the rate R and whatever the memory ν. Thus, at low SNR's, the BER of the RSC code is always smaller than the BER of the equivalent NSC code.

In general, for rates $R \leq 2/3$, the first coefficients $(W_{\mathrm{RSC}}(d_f)), W_{\mathrm{RSC}}(d_f + 1))$ are larger than those of NSC codes, therefore, at large SNR's, the performance of NSC codes is a little better than that of RSC codes. When the rate is larger than 2/3, it is easy to find RSC codes whose performance is better than that of NSC codes at any SNR.

In order to illustrate the performance of RSC codes, the BER of RSC and NSC codes are plotted in Fig. 3 for different values of R and for an encoder with memory $\nu = 6$ and generators 133, 171. For instance, at $P_e = 10^{-1}$, the coding gain with this RSC code, relative to the equivalent NSC code, is approximately 0.8 dB at $R = 2/3$, whereas at $R = 3/4$, it reaches 1.75 dB.

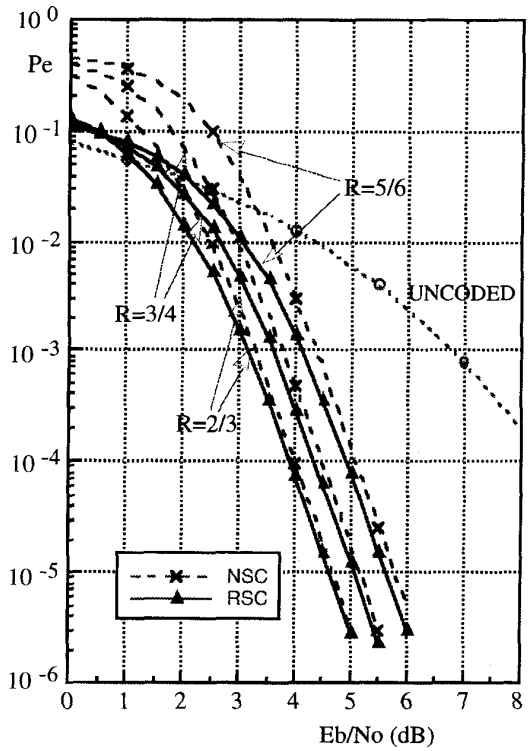


Fig. 3. P_e of punctured RSC and NSC codes for different values of rate R and memory $\nu = 6$, generators $G_1 = 133, G_2 = 171$.

III. Parallel Concatenation with Non Uniform Interleaving: Turbo-Code

A. Construction of the Code

The use of systematic codes enables the construction of a concatenated encoder in the form given in Fig. 4, called *parallel concatenation.* The data flow (d_k at time k) goes directly to a first elementary RSC encoder C_1 and after interleaving, it feeds (d_n at time k) a second elementary RSC encoder C_2. These two encoders are not necessarily identical. Data d_k is systematically transmitted as symbol X_k and redundancies Y_{1k} and Y_{2k} produced by C_1 and C_2 may be completely transmitted for an $R = 1/3$ encoding or punctured for higher rates. The two elementary coding rates R_1 and R_2 associated with C_1 and C_2, after puncturing, may be different, but for the best decoding performance, they will satisfy$R_1 \leq R_2$. The global rate R of the composite code, R_1 and R_2 are linked by

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} - 1. \tag{13}$$

Unlike the classical (serial) concatenation, parallel concatenation enables the elementary encoders, and therefore the associated elementary decoders, to run with the same clock. This point constitutes an important simplification for the design of the associated circuits, in a concatenated scheme.

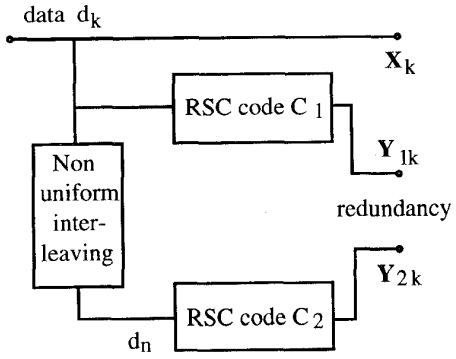


Fig. 4. Basic turbo-encoder (rate 1/3).

B. Distance Properties

Consider for instance elementary codes C_1 and C_2 with memory $\nu = 4$ and encoder polynomials $G_1 = 23, G_2 = 35$. Redundancy Y_k is one time every second time, either Y_{1k} or Y_{2k}. Then the global rate is $R = 1/2$ with $R_1 = R_2 = 2/3$. The code being linear, the distance properties will be considered relatively to the "all zero" or "null" sequence. Both encoders C_1, C_2 and the interleaver are initialized to the "all zero" state and a sequence $\{d_k\}$ containing w "1"s, is fed to the turbo-encoder. w is called the *input weight.*

Some definitions: let us call a *Finite Codeword* (FC) of an elementary RSC encoder an output sequence with a finite distance from the "all zero" sequence (i.e., a limited number of "1"s in output sequences $\{X_k\}$ and $\{Y_k\}$). Because of recursivity, only some input sequences, fitting with the linear feedback register (LFR) generation of Y_k, give FC's. These particular input sequences are named *FC (input) patterns.* Let us also call a *global FC* an output sequence of the turbo-code, with a finite distance from the "all zero" sequence (i.e., a limited number of "1"s in output sequences $\{X_k\}, \{Y_{1k}\}$ and $\{Y_{2k}\}$). The distance $d_q(w)$ of an elementary FC ($q = 1$ for $C_1, q = 2$ for C_2), associated with an input sequence $\{d_k\}$ with weight w, is the sum of the two contributions of $\{X_k\}$ and $\{Y_{qk}\}$

$$d_q(w) = d_{Xq}(w) + d_{Yq}(w). \tag{14}$$

Since the codes are systematic, $d_{Xq}(w) = w$

$$d_q(w) = w + d_{Yq}(w). \tag{15}$$

The distance $d(w)$ of a global FC is given likewise by

$$d(w) = w + d_{Y1}(w) + d_{Y2}(w). \tag{16}$$

1) Uniform Interleaving: Consider a uniform block interleaving using an $M \cdot M$ square matrix with M large enough (i.e., ≥ 32), and generally a power of 2. Data are written linewise and read columnwise. As explained above, the matrix is filled with "0"s except for some "1"s and we are now going to state some of the possible patterns of "1"s leading to global FC's and evaluate their associated distances. Beforehand, note that, for each elementary code, the minimal value for input weight w is two, because of their recursive structure. For the particular case of $w = 2$, the delay between the two data at "1" is 15 or a multiple of 15, since the LFR associated with the redundancy generation, with polynomial $G = 23$, is a maximum length LFR. If the delay is not a multiple of 15, then the associated sequence $\{Y_k\}$ will contain "0"s and "1"s indefinitely.

Global FC's with Input Weight $w = 2$

Global FC's with $w = 2$ have to be FC's with input weight $w = 2$ for each of the constituent codes. Then the two data at "1" in the interleaving memory must be located at places which are distant by a multiple of 15, when both writing and reading. For lack of an extensive analysis of the possible patterns which satisfy this double constraint, let us consider only the case where the two "1"s in $\{d_k\}$ are time-separated by the lowest value (i. e. 15). It is also a FC pattern for code C_2 if the two data at "1" are memorized on a single line, when writing in the interleaving memory, and thus the span between the two "1"s is $15 \cdot M$, when reading. From (16), the distance associated with this input pattern is approximately equal to

$$\begin{aligned} d(2) &\approx 2 + \min\{d_{Y1}(2)\} + \mathrm{INT}((15 \cdot M + 1)/4) \\ &= 2 + 4 + \mathrm{INT}((15 \cdot M + 1)/4) \end{aligned} \tag{17}$$

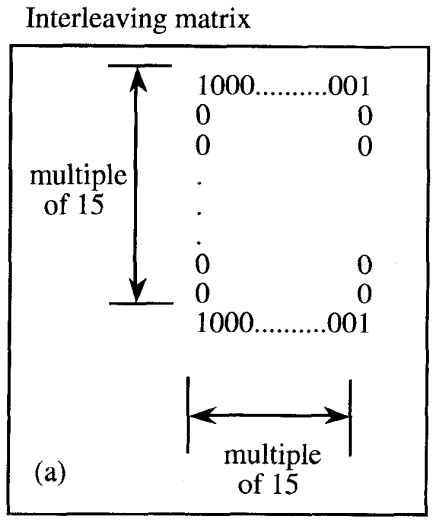
where $\mathrm{INT}(\cdot)$ stands for integer part of $(\cdot)$. The first term represents input weight w, the second term the distance added by redundancy Y_1 of C_1 (rate 2/3), the third term the distance added by redundancy Y_2. The latter is given assuming that, for this second 2/3 rate code, the $(15 \cdot M + 1)/2$ Y_2 symbols are at "1", one out of two times statistically, which explains the additional division by 2. With $M = 64$ for instance, the distance is $d(2) \approx 246$. This value is for a particular case of a pattern of two "1"s but we imagine it is a realistic example for all FC's with input weight 2. If the two "1"s are not located on a single line when writing in the interleaving matrix, and if M is larger than 30, the span between the two "1"s, when reading from the interleaving matrix, is higher than $15 \cdot M$ and the last term in (17) is increased.

Global FC's with $w = 3$

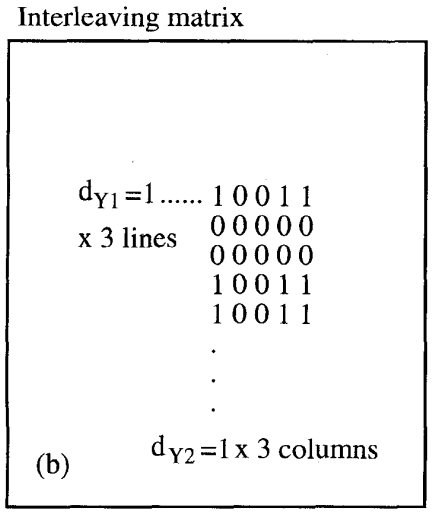
Global FC's with input weight $w = 3$ have to be elementary FC's with input weight $w = 3$ for each of the constituent codes. The inventory of the patterns with three "1"s which satisfy this double constraint is not easy to make. It is not easier to give the slightest example. Nevertheless, we can consider that associated distances are similar to the case of input weight two codewords, because the FC for C_2 is still several times M long.

Global FC's with $w = 4$

Here is the first pattern of a global FC which can be viewed as the separate combination of two minimal (input weight $w = 2$) elementary FC patterns both for C_1 and C_2. When writing in the interleaving memory, the four data at "1" are located at the four corners of a square or a rectangle, the sides of which have lengths equal to 15 or a multiple of 15 [Fig. 5(a)]. The minimum distance is given by a square pattern, with side length equal to 15, corresponding to minimum values



(a)



(b)

Fig. 5. (a) Input pattern for separable global FC's with input weight $w = 4$. (b) Input pattern for a global FC with $w = 9$ and total distance of 15.

of $d_{Y1}(2)$ and $d_{Y2}(2)$

$$d(4) = 4 + 2.\min\{d_{Y1}(2)\} + 2.\min\{d_{Y2}(2)\} = 4 + 8 + 8 = 20. \quad (18)$$

Global FC's with $w > 4$

As previously, one has to distinguish between cases where the global FC is separable into two, or more, elementary FC's for both codes, or not. The shortest distances are given by separable codewords (this occurs for weights 6, 8, 9, 10, $12, \cdots$). We made an exhaustive research of possible patterns up to $w = 10$ and found a minimum distance of 15 for the pattern given in Fig. 5(b), with $w = 9$, which corresponds to three separable FC's for each of the two codes, and the six FC's having d_{Y1} or d_{Y2} equal to 1.

To conclude this review of the global FC's with the lowest values of we can retain the result obtained for the case in Fig. 5(b) as the minimum distance d_m of this particular turbo-code ($\nu = 4$, polynomials 23, 35, $R_1 = R_2 = 2/3$). Other codes were considered with different values of ν and various polynomials; in all cases, the minimum distance seems to correspond to input sequences with weights 4, 6, or 9. The values of d_m are within 10 and 20, which is not a remarkable property. So as to obtain larger values of d_m, turbo-coding employs nonuniform interleaving.

2) Nonuniform interleaving: It is obvious that patterns giving the shortest distances, such as those represented in Fig. 5, can be "broken" by appropriate nonuniform interleaving, in order to transform a separable FC pattern into either a nonseparable or a non FC. Nonuniform interleaving must satisfy two main conditions: the maximum scattering of data, as in usual interleaving, and the maximum disorder in the interleaved data sequence. The latter, which may be in conflict with the former, is to make redundancy generation by the two encoders as diverse as possible. In this case, if the decision by the decoder associated with C_1 about particular data implies a few items of redundancy Y_1, then the corresponding decision by the decoder associated with C_2 will rely on a large number of values Y_2, and vice-versa. Then, the minimum distance of the turbo-code may be increased to much larger values than that given by uniform interleaving.

The mathematical aspects of nonuniform interleaving are not trivial and have still to be studied. For instance, how can it be ensured that an interleaver, able to "break" patterns corresponding to $w = 4, 6$, or 9, will not drastically lower the distance for a particular case of a $w = 2$ input sequence? On the other hand, the interleaving equations have to be limited in complexity for silicon applications, because several interleavers and de-interleavers have to be employed in a real-time turbo-decoder (see Section VI-A). However that may be, as we have tried to explain up to now, one important property of the turbo-code is that its minimum distance d_m is not fixed, chiefly, by the constituent RSC codes but by the interleaving function; and finding out the optimum interleaver for turbo-codes remains a real challenge.

For the results which are presented in Section VII, we used an empirical approach and chose an interleaving procedure in which, for reading, the column index is a function of the line index. Let i and j be the addresses of line and column for writing, and i_r and j_r the addresses of line and column for reading. For an $M \cdot M$ memory (M being a power of two), i, j, i_r and j_r have values between 0 and $M - 1$. Nonuniform interleaving may be described by

$$\begin{aligned} i_r &= (M/2 + 1)(i + j) && \mod \cdot M \\ \xi &= (i + j) && \mod \cdot 8 \\ j_r &= [P(\xi) \cdot (j + 1)] - 1 && \mod \cdot M \end{aligned} \quad (19)$$

$P(\cdot)$ is a number, relatively prime with M, which is a function of line address $(i + j) \bmod \cdot 8$. Note that reading is performed diagonally in order to avoid possible effects of a relation between M and the period of puncturing. A multiplying factor $(M/2 + 1)$ is used to prevent two neighboring data written on two consecutive lines from remaining neighbors in reading, in a similar way as given in [9].

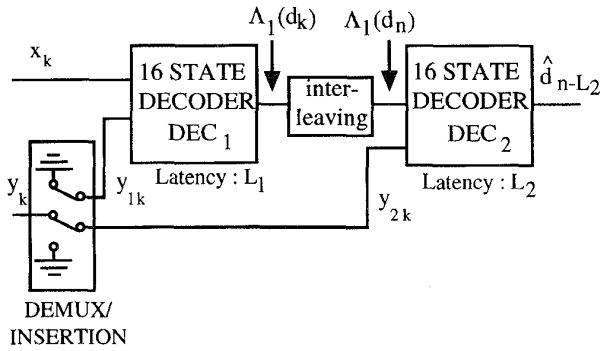


Fig. 6. Principle of the decoder in accordance with a serial concatenation scheme.

IV. Decoding Turbo-Codes

The decoder depicted in Fig. 6, is made up of two elementary decoders (DEC_1 and DEC_2) in a serial concatenation scheme. The first elementary decoder DEC_1 is associated with the lower rate R_1 encoder C_1 and yields a weighted decision.

For a discrete memoryless Gaussian channel and a binary modulation, the decoder input is made up of a couple R_k of two random variables x_k and y_k, at time k

$$\begin{aligned} x_k &= (2X_k - 1) + i_k \\ y_k &= (2Y_k - 1) + q_k \end{aligned} \tag{20}$$

where i_k and q_k are two independent noises with the same variance σ^2. The redundant information y_k is demultiplexed and sent to decoder DEC_1 when $Y_k = Y_{1k}$ and toward decoder DEC_2 when $Y_k = Y_{2k}$. When the redundant information of a given encoder (C_1 or C_2) is not emitted, the corresponding decoder input is set to analog zero. This is performed by the DEMUX/INSERTION block.

The logarithm of likelihood ratio (LLR), $\Lambda_1(d_k)$ associated with each decoded bit d_k by the first decoder DEC_1 can be used as a relevant piece of information for the second decoder DEC_2

$$\Lambda_1(d_k) = \log \frac{\Pr\{d_k = 1/\text{observation}\}}{\Pr\{d_k = 0/\text{observation}\}} \tag{21}$$

where $\Pr\{d_k = i/\text{observation}\}, i = 0, 1$ is the *a posteriori* probability (APP) of the bit d_k.

A. Optimal Decoding of RSC Codes with Weighted Decision

The VITERBI algorithm is an optimal decoding method which minimizes the probability of sequence error for convolutional codes. Unfortunately, this algorithm is not able to yield directly the APP for each decoded bit. A relevant algorithm for this purpose has been proposed by BAHL *et al.* [10]. This algorithm minimizes the bit error probability in decoding linear block and convolutional codes and yields the APP for each decoded bit. For RSC codes, the BAHL *et al.* algorithm must be modified in order to take into account their recursive character.

Modified BAHL et al. Algorithm for RSC Codes: Consider an RSC code with constraint length K; at time k the encoder state $\boldsymbol{S}_k$ is represented by a K-uple

$$\boldsymbol{S}_k = (a_k, a_{k-1} \cdots\cdots a_{k-K+2}). \tag{22}$$

Let us also suppose that the information bit sequence $\{d_k\}$ is made up of N independent bits d_k, taking values zero and one with equal probability and that the encoder initial state $\boldsymbol{S}_0$ and final state $\boldsymbol{S}_N$ are both equal to zero, *i.e*

$$\boldsymbol{S}_0 = \boldsymbol{S}_N = (0, 0 \cdots\cdots 0) = \boldsymbol{0}. \tag{23}$$

The encoder output codeword sequence, noted $\boldsymbol{C}_1^N = \{C_1 \cdots\cdots C_k \cdots\cdots C_N\}$ where $C_k = (X_k, Y_k)$ is the input to a discrete Gaussian memoryless channel whose output is the sequence $\boldsymbol{R}_1^N = \{R_1 \cdots\cdots R_k \cdots\cdots R_N\}$ where $R_k = (x_k, y_k)$ is defined by (20).

The APP of a decoded bit d_k can be derived from the joint probability $\lambda_k^i(\boldsymbol{m})$ defined by

$$\lambda_k^i(\boldsymbol{m}) = \Pr\{d_k = i, \boldsymbol{S}_k = \boldsymbol{m}/\boldsymbol{R}_1^N\} \tag{24}$$

and thus, the APP of a decoded bit d_k is equal to

$$\Pr\{d_k = i/\boldsymbol{R}_1^N\} = \sum_{\boldsymbol{m}} \lambda_k^i(\boldsymbol{m}), \qquad i = 0, 1. \tag{25}$$

From (21) and (24), the LLR $\Lambda(d_k)$ associated with a decoded bit d_k can be written as

$$\Lambda(d_k) = \log \frac{\sum_{\boldsymbol{m}} \lambda_k^1(\boldsymbol{m})}{\sum_{\boldsymbol{m}} \lambda_k^0(\boldsymbol{m})}. \tag{26}$$

Finally the decoder can make a decision by comparing $\Lambda(d_k)$ to a threshold equal to zero

$$\begin{aligned} \hat{d}_k &= 1 \quad \text{if} \quad \Lambda(d_k) \geq 0 \\ \hat{d}_k &= 0 \quad \text{if} \quad \Lambda(d_k) < 0. \end{aligned} \tag{27}$$

From the definition (24) of $\lambda_k^i(\boldsymbol{m})$, the LLR $\Lambda(d_k)$ can be written as shown at the bottom of the page.

Using the BAYES rule and taking into account that events after time k are not influenced by observation $\boldsymbol{R}_1^k$ and bit d_k

$$\Lambda(d_k) = \log \frac{\sum_{\boldsymbol{m}} \sum_{\boldsymbol{m}'} \Pr\{d_k = 1, \boldsymbol{S}_k = \boldsymbol{m}, \boldsymbol{S}_{k-1} = \boldsymbol{m}', \boldsymbol{R}_1^{k-1}, R_k, R_{k+1}^N\}}{\sum_{\boldsymbol{m}} \sum_{\boldsymbol{m}'} \Pr\{d_k = 0, \boldsymbol{S}_k = \boldsymbol{m}, \boldsymbol{S}_{k-1} = \boldsymbol{m}', \boldsymbol{R}_1^{k-1}, R_k, R_{k+1}^N\}} \tag{28}$$

if state S_k is known, the LLR $\Lambda(d_k)$ is equal

$$\Lambda(d_k) = \log \frac{\sum_{m}\sum_{m'} \Pr\{R_{k+1}^N/S_k = m\}\Pr\{S_{k-1} = m'/R_1^{k-1}\}}{\sum_{m}\sum_{m'} \Pr\{R_{k+1}^N/S_k = m\}\Pr\{S_{k-1} = m'/R_1^{k-1}\}} \cdot \frac{\Pr\{d_k = 1, S_k = m, R_k/S_{k-1} = m'\}}{\Pr\{d_k = 0, S_k = m, R_k/S_{k-1} = m'\}}. \quad (29)$$

In order to compute the LLR $\Lambda(d_k)$, as in [11] let us introduce the probability functions $\alpha_k(m), \beta_k(m)$ and $\gamma_i(R_k, m', m)$ defined by

$$\alpha_k(m) = \Pr\{S_k = m/R_1^k\} \quad (30a)$$

$$\beta_k(m) = \frac{\Pr\{R_{k+1}^N/S_k = m\}}{\Pr\{R_{k+1}^N/R_1^k\}} \quad (30b)$$

$$\gamma_i(R_k, m', m) = \Pr\{d_k = i, S_k = m, R_k/S_{k-1} = m'\}. \quad (30c)$$

Using the LLR definition (29) and (30a), (30b) and (30c), $\Lambda(d_k)$ is equal to

$$\Lambda(d_k) = \log \frac{\sum_{m}\sum_{m'} \gamma_1(R_k, m', m)\alpha_{k-1}(m')\beta_k(m)}{\sum_{m}\sum_{m'} \gamma_0(R_k, m', m)\alpha_{k-1}(m')\beta_k(m)} \quad (31)$$

where probabilities $\alpha_k(m)$ and $\beta_k(m)$ can be recursively calculated from probability $\gamma_i(R_k, m', m)$ as in [12]

$$\alpha_k(m) = \frac{\sum_{m'}\sum_{i=0}^{1} \gamma_i(R_k, m', m)\alpha_{k-1}(m')}{\sum_{m}\sum_{m'}\sum_{i=0}^{1} \gamma_i(R_k, m', m)\alpha_{k-1}(m')} \quad (32)$$

$$\beta_k(m) = \frac{\sum_{m'}\sum_{i=0}^{1} \gamma_i(R_{k+1}, m', m)\beta_{k+1}(m')}{\sum_{m}\sum_{m'}\sum_{i=0}^{1} \gamma_i(R_{k+1}, m, m')\alpha_k(m')}. \quad (33)$$

The probability $\gamma_i(R_k, m', m)$ can be determined from transition probabilities of the discrete Gaussian memoryless channel and transition probabilities of the encoder trellis. From (30c), $\gamma_i(R_k, m', m)$ is given by

$$\gamma_i(R_k, m', m) = p(R_k/d_k = i, S_k = m, S_{k-1} = m') \cdot q(d_k = i/S_k = m, S_{k-1} = m') \cdot \pi(S_k = m/S_{k-1} = m') \quad (34)$$

where $p(\cdot/\cdot)$ is the transition probability of the discrete Gaussian memoryless channel. Conditionally to $(d_k = i, S_k = m, S_{k-1} = m'), x_k$ and y_k $(R_k = (x_k, y_k))$ are two uncorrelated Gaussian variables and thus we obtain

$$p(R_k/d_k = i, S_k = m, S_{k-1} = m') = p(x_k/d_k = i, S_k = m, S_{k-1} = m') \cdot p(y_k/d_k = i, S_k = m, S_{k-1} = m'). \quad (35)$$

Since the convolutional encoder is a deterministic machine, $q(d_k = i/S_k = m, S_{k-1} = m')$ is equal to 0 or 1. The transition state probabilities $\pi(S_k = m/S_{k-1} = m')$ of the trellis are defined by the encoder input statistic. Generally, $\Pr\{d_k = 1\} = \Pr\{d_k = 0\} = 1/2$ and since there are two possible transitions from each state, $\pi(S_k = m/S_{k-1} = m') = 1/2$ for each of these transitions.

Modified BAHL et al. Algorithm:

Step 0: Probabilities $\alpha_0(m)$ are initialized according to condition (23)

$$\alpha_0(0) = 1; \quad \alpha_0(m) = 0 \quad \forall m \neq 0. \quad (36a)$$

Concerning the probabilities $\beta_N(m)$, the following conditions were used

$$\beta_N(m) = \frac{1}{N} \forall m \quad (36b)$$

since it is very difficult to put the turbo encoder at state 0, at a given time. Obviously with this condition, the last bits of each block are not taken into account for evaluating the BER.

Step 1: For each observation R_k, probabilities $\alpha_k(m)$ and $\gamma_i(R_k, m', m)$ are computed using (32) and (34), respectively.

Step 2: When sequence R_1^N has been completely received, probabilities $\beta_k(m)$ are computed using (33), and the LLR associated with each decoded bit d_k is computed from (31).

V. Extrinsic Information from the Decoder

In this chapter, we will show that the LLR $\Lambda(d_k)$ associated with each decoded bit d_k, is the sum of the LLR's of d_k at the decoder input and of other information called *extrinsic information,* generated by the decoder.

Since the encoder is systematic $(X_k = d_k)$, the transition probability $p(x_k/d_k = i, S_k = m, S_{k-1} = m')$ in expression $\gamma_i(R_k, m', m)$ is independent of state values S_k and S_{k-1}. Therefore we can factorize this transition probability in the numerator and in the denominator of (31)

$$\Lambda(d_k) = \log \frac{p(x_k/d_k = 1)}{p(x_k/d_k = 0)} + \log \frac{\sum_{m}\sum_{m'} \gamma_1(y_k, m', m)\alpha_{k-1}(m')\beta_k(m)}{\sum_{m}\sum_{m'} \gamma_0(y_k, m', m)\alpha_{k-1}(m')\beta_k(m)}. \quad (37)$$

Conditionally to $d_k = 1$ (resp. $d_k = 0$), variables x_k are Gaussian with mean 1 (resp. -1) and variance σ^2, thus the LLR $\Lambda(d_k)$ is still equal to

$$\Lambda(d_k)\frac{2}{\sigma^2} x_k + W_k \quad (38)$$

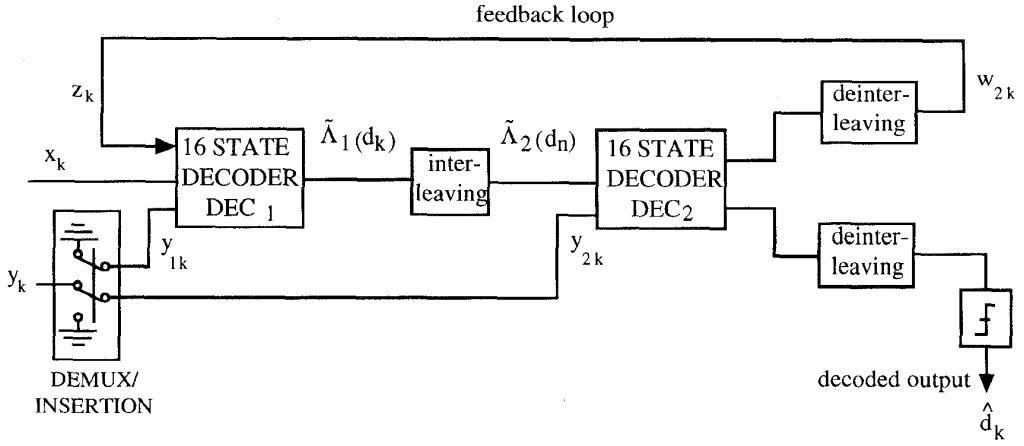


Fig. 7. Feedback decoder (under 0 internal delay assumption).

where

$$W_k = \Lambda(d_k)\,|_{x_k=0} = \log \frac{\sum_{\boldsymbol{m}} \sum_{\boldsymbol{m}'} \gamma_1(y_k, \boldsymbol{m}', \boldsymbol{m}) \alpha_{k-1}(\boldsymbol{m}') \beta_k(\boldsymbol{m})}{\sum_{\boldsymbol{m}} \sum_{\boldsymbol{m}'} \gamma_0(y_k, \boldsymbol{m}', \boldsymbol{m}) \alpha_{k-1}(\boldsymbol{m}') \beta_k(\boldsymbol{m})}. \quad (39)$$

W_k is a function of the redundant information introduced by the encoder. In general, W_k has the same sign as d_k; therefore W_k may improve the LLR associated with each decoded bit d_k. This quantity represents the extrinsic information supplied by the decoder and does not depend on decoder input x_k. This property will be used for decoding the two parallel concatenated encoders.

VI. Decoding Scheme of Parallel Concatenation Codes

In the decoding scheme represented in Fig. 6, decoder DEC_1 computes LLR $\Lambda_1(d_k)$ for each transmitted bit d_k from sequences $\{x_k\}$ and $\{y_k\}$, then decoder DEC_2 performs the decoding of sequence $\{d_k\}$ from sequences $\{\Lambda_1(d_k)\}$ and $\{y_k\}$. Decoder DEC_1 uses the modified BAHL *et al.* algorithm and decoder DEC_2 may use the VITERBI algorithm. The global decoding rule is not optimal because the first decoder uses only a fraction of the available redundant information. Therefore it is possible to improve the performance of this serial decoder by using a feedback loop.

A. Decoding with a Feedback Loop

Both decoders DEC_1 and DEC_2 now use the modified BAHL *et al.* algorithm. We have seen in section V that the LLR at the decoder output can be expressed as a sum of two terms if the noises at the decoder inputs are independent at each time k. Hence, if the noise at the decoder DEC_2 inputs are independent, the LLR $\Lambda_2(d_k)$ at the decoder DEC_2 output can be written as

$$\Lambda_2(d_k) = f(\Lambda_1(d_k)) + W_{2k} \quad (40)$$

with

$$\Lambda_1(d_k) = \frac{2}{\sigma^2} x_k + W_{1k}. \quad (41)$$

From (39), we can see that the decoder DEC_2 extrinsic information W_{2k} is a function of the sequence $\{\Lambda_1(d_n)\}_{n \neq k}$. Since $\Lambda_1(d_n)$ depends on observation $\boldsymbol{R}_1^N$, extrinsic information W_{2k} is correlated with observations x_k and y_{1k}, regarding the noise. Nevertheless, from (39), the greater $|n-k|$ is, the less correlated are $\Lambda_1(d_n)$ and observations x_k, y_k. Thus, due to the presence of interleaving between decoders DEC_1 and DEC_2, extrinsic information W_{2k} and observations x_k, y_{1k} are weakly correlated. Therefore extrinsic information W_{2k} and observations x_k, y_{1k} can be used jointly for carrying out a new decoding of bit d_k, the extrinsic information $z_k = W_{2k}$ acting as a diversity effect.

In Fig. 7, we have depicted a new decoding scheme using the extrinsic information W_{2k} generated by decoder DEC_2 in a feedback loop. For simplicity this drawing does not take into account the different delays introduced by decoder DEC_1 and DEC_2, and interleaving.

The first decoder DEC_1 now has three data inputs: (x_k, y_k^1, z_k), and probabilities $\alpha_{1k}(\boldsymbol{m})$ and $\beta_{1k}(\boldsymbol{m})$ are computed by substituting $R_k = (x_k, y_{1k}, z_k)$ for $R_k = (x_k, y_{1k})$ in (32) and (33). Taking into account that z_k is weakly correlated with x_k and y_{1k} and supposing that z_k can be approximated by a Gaussian variable with variance $\sigma_z^2 \neq \sigma^2$, the transition probability of the discrete gaussian memoryless channel can be now factorized in three terms

$$p(R_k/d_k = i, \boldsymbol{S}_k = \boldsymbol{m}, \boldsymbol{S}_{k-1} = \boldsymbol{m}') = p(x_k/\cdot) p(y_k^1/\cdot) p(z_k/\cdot). \quad (42)$$

Encoder C_1 with initial rate R_1, through the feedback loop, is now equivalent to a rate R_1' encoder with

$$R_1' = \frac{R_1}{1 + R_1}. \quad (43)$$

The first decoder obtains additional redundant information with Z_k that may significantly improve its performance ; the term turbo-code is given for this feedback decoder scheme with reference to the principle of the turbo engine.

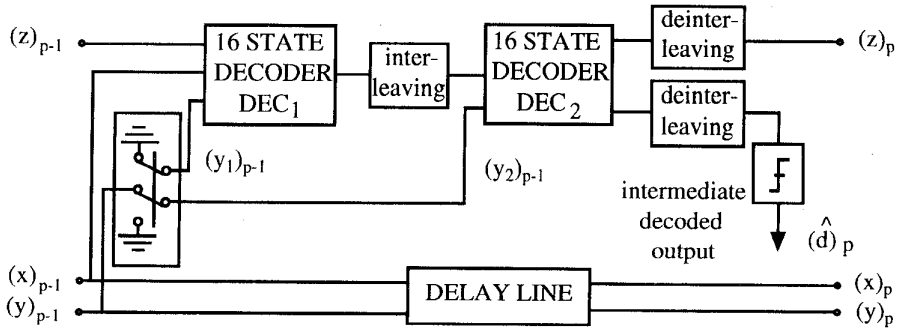


Fig. 8. Decoding module (level p).

With the feedback decoder, the LLR $\Lambda_1(d_k)$ generated by decoder DEC_1 is now equal to

$$\Lambda_1(d_k) = \frac{2}{\sigma^2} x_k + \frac{2}{\sigma_z^2} z_k + W_{1k} \tag{44}$$

where W_{1k} depends on sequence $\{z_n\}_{n \neq k}$. As indicated above, information z_k has been built by decoder DEC_2. Therefore z_k must not be used as input information for decoder DEC_2. Thus decoder DEC_2 input sequences will be sequences $\{\tilde{\Lambda}_1(d_n)\}$ and $\{y_{2k}\}$ with

$$\tilde{\Lambda}_1(d_n) = \Lambda_1(d_n)_{z_n=0}. \tag{45}$$

Finally, from (40), decoder DEC_2 extrinsic information $z_k = W_{2k}$ after deinterleaving can be written as

$$z_k = W_{2k} = \Lambda_2(d_k)|_{\tilde{\Lambda}_1(d_k)=0} \tag{46}$$

and the decision at the decoder DEC output is

$$\hat{d}_k = \mathrm{sign}[\Lambda_2(d_k)]. \tag{47}$$

The decoding delays introduced by the component decoders, the interleaver and the deinterleaver imply that the feedback piece of information z_k must be used through an iterative process.

The global decoder circuit is made up of P pipelined identical elementary decoders. The pth decoder DEC (Fig. 8) input, is made up of demodulator output sequences $(x)_p$ and $(y)_p$ through a delay line and of extrinsic information $(z)_p$ generated by the $(p-1)$th decoder DEC. Note that the variance σ_z^2 of $(z)_p$ and the variance of $\tilde{\Lambda}_1(d_k)$ must be estimated at each decoding step p.

For example, the variance σ_z^2 is estimated for each M^2 interleaving matrix by the following:

$$\sigma_z^2 = \frac{1}{M^2} \sum_{k=1}^{M^2} (|z_k| - m_z)^2 \tag{48a}$$

where m_z is equal to

$$m_z = \frac{1}{M^2} \sum_{k=1}^{M^2} |z_k|. \tag{48b}$$

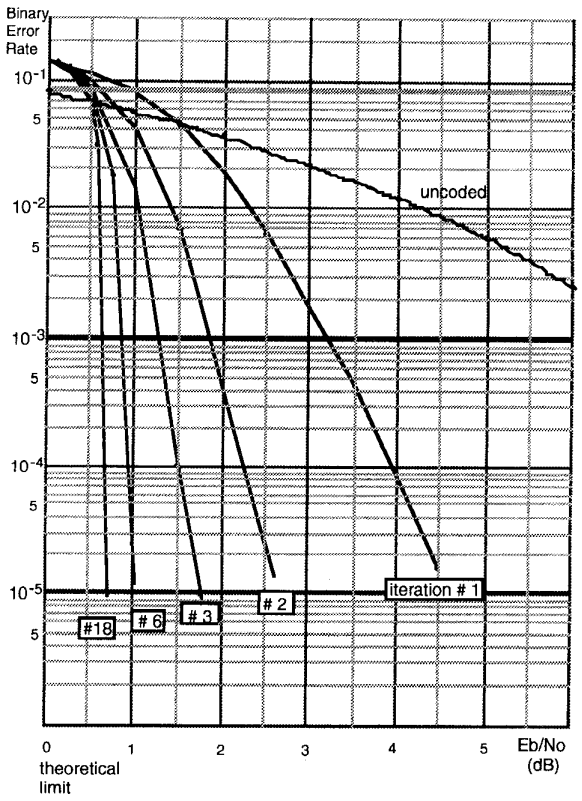


Fig. 9. BER given by iterative decoding ($p = 1, \cdots 18$) of a rate $R = 1/2$ encoder, memory $\nu = 4$, generators $G_1 = 37, G_2 = 21$, with interleaving 256×256.

B. Interleaving

The interleaver is made up of an $M \cdot M$ matrix and bits $\{d_k\}$ are written row by row and read following the nonuniform rule given in Section III-B2. This nonuniform reading procedure is able to spread the residual error blocks of rectangular form, that may set up in the interleaver located behind the first decoder DEC_1, and to give a large free distance to the concatenated (parallel) code.

For the simulations, a 256.256 matrix has ben used and from (19), the addresses of line i_r and column j_r for reading are the following:

$$i_r = 129(i + j) \qquad \mathrm{mod} \cdot 256$$

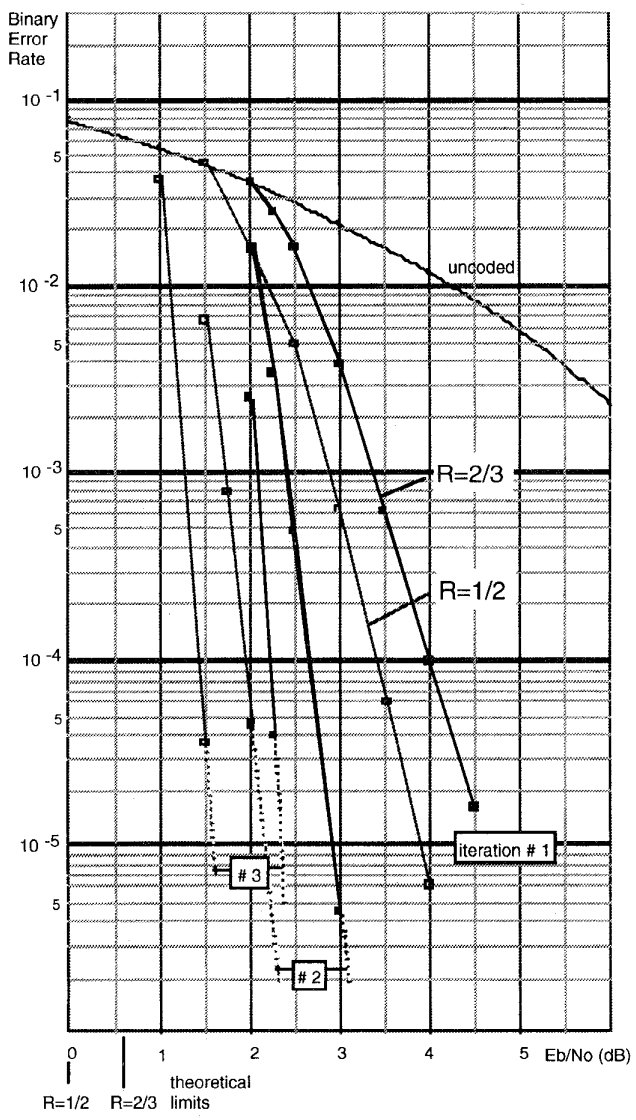


Fig. 10. BER given by iterative decoding ($p = 1, 2, 3$) of code with memory $\nu = 4$, generators $G_1 = 23, G_2 = 35$, rate $R = 1/2$ and $R = 2/3$ and interleaving 256×256 decoding.

$$\begin{aligned} \xi &= (i + j) && \mod \cdot\, 8 \\ j_r &= [P(\xi) \cdot (j + 1)] - 1 && \mod \cdot\, 256 \end{aligned} \tag{49a}$$

with

$$\begin{aligned} &P(0) = 17; \quad P(1) = 37; \quad P(2) = 19; \quad P(3) = 29 \\ &P(4) = 41; \quad P(5) = 23; \quad P(6) = 13; \quad P(7) = 7. \end{aligned} \tag{49b}$$

VII. Results

For a rate $R = 1/2$ encoder with memory $\nu = 4$, and for generators $G_1 = 37, G_2 = 21$ and parallel concatenation 2/3//2/3 ($R_1 = 2/3, R_2 = 2/3$), the BER has been computed after each decoding step using the Monte Carlo method, as a function of signal to noise ratio E_b/N_0. The interleaver consists of a 256×256 matrix and the modified BAHL *et al.* algorithm has been used with a data block length of $N = 65\,536$ bits. In order to evaluate a BER equal to 10^{-5}, we have considered 256 data blocks *i.e.* approximately 16×10^6 b d_k. The BER versus E_b/N_0, for different values of p is plotted in Fig. 9. For any given SNR greater than 0 dB, the BER decreases as a function of the decoding step p. The coding gain is fairly high for the first values of $p (p = 1, 2, 3)$ and carries on increasing for the subsequent values of p. For $p = 18$ for instance, the BER is lower than 10^{-5} at $E_b/N_0 = 0.7$ dB with generators $G_1 = 37, G_2 = 21$. Remember that the Shannon limit for a binary modulation is $P_e = 0$ (several authors take $P_e = 10^{-5}$ as a reference) for $E_b/N_0 = 0$ dB. With the parallel concatenation of RSC convolutional codes and feedback decoding, the performance is 0.7 dB from Shannon's limit. The influence of the memory ν on the BER has also been examined. For memory greater than 4, at $E_b/N_0 = 0.7$ dB, the BER is slightly worse at the first ($p = 1$) decoding step and the feedback decoding is inefficient to improve the final BER. For ν smaller than 4, at $E_b/N_0 = 0.7$ dB, the BER is slightly better at the first decoding step than for ν equal to four, but the correction capacity of encoders C_1 and C_2 is too weak to improve the BER with feedback decoding. For $\nu = 3$ (i.e., with only eight-states decoders) and after iteration 18, a BER of 10^{-5} is achieved at $E_b/N_0 = 0.9$ dB.

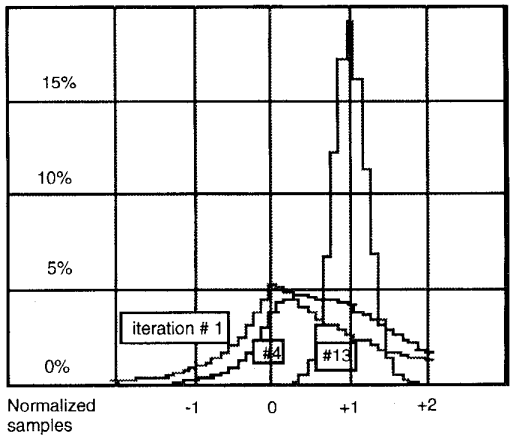


Fig. 11. Histograms of extrinsic information z_k after iterations #1, 4, 13 at $E_b/N_0 = 0.8$ dB; all information bits $d_k = 1$.

For ν equal to 4, we have tested several generators (G_1, G_2) and the best results were achieved with $G_1 = 37, G_2 = 21$ and at least six iterations ($p \geq 6$). For p smaller than four, generators $G_1 = 23, G_2 = 35$ lead to better performance than generators $G_1 = 37, G_2 = 21$, at large SNR's (Fig. 10). This result can be understood since, with generators $G_1 = 23, G_2 = 35$, the first coefficients $W_{RSC}(d)$ are smaller than with generators $G_1 = 37, G_2 = 21$. For very low SNR's, the BER can sometimes increase during the iterative decoding process. In order to overcome this effect the extrinsic information z_k has been divided by $[1 + \theta|\tilde{\Lambda}_1(d_k)|]$. θ acts as a stability factor and its value of 0.15 was adopted after several simulation tests at $E_b/N_0 = 0.7$ dB.

We have also plotted the BER for a rate $R = 2/3$ encoder with memory $\nu = 4$, generators $G_1 = 23, G_2 = 35$ and

parallel concatenation 4/5//4/5 in Fig. 10. For a nonuniform interleaver consisting of a 256×256 matrix and for $p = 3$, the BER is equal to 10^{-5} at $E_b/N_0 = 1.6$ dB and thus the performance is only 1 dB from Shannon's limit.

In Fig. 11, the histogram of extrinsic information $(z)_p$ (generators $G_1 = 37, G_2 = 21.$) has been drawn for several values of iteration p, with all bits equal to one and for a low SNR ($E_b/N_0 = 0.8$ dB). For $p = 1$ (first iteration), extrinsic information $(z)_p$ is very poor about bit d_k, and furthermore, the gaussian hypothesis made above for extrinsic information $(z)_p$, is not satisfied! Nevertheless, when iteration p increases, the histogram merges toward a Gaussian law with a mean equal to one, after normalization. For instance, for $p = 13$, extrinsic information $(z)_p$ becomes relevant information concerning bits d_k.

Note that the concept of turbo-codes has been recently extended to block codes, in particular by Pyndiah *et al.* [13].

ACKNOWLEDGMENT

The authors wish to express their grateful acknowledgment to D. Divsalar for his constructive remarks about the organization of this paper.

REFERENCES

[1] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE,* vol. 61, no. 3, pp. 268–278, Mar. 1973.
[2] J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured convolutional codes of rate (n-1)/n and simplified maximum likelihood decoding," *IEEE Trans. Inform. Theory,* vol. IT-25, pp. 97–100, Jan 1979.
[3] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory,* vol. IT-28, no. 1, pp. 55–67, Jan. 1982.
[4] Y. Yasuda, K. Kashiki, and Y. Hirata, "High-rate punctured convolutional codes for soft-decision Viterbi decoding," *IEEE Trans. Commun.,* vol. COM-32, no. 3, Mar. 1984.
[5] A. J. Viterbi and J. K. Omura. *Principles of digital communication and coding.* New York: MacGraw-Hill, 1979.
[6] A. J. Viterbi, "Convolutional codes and their performance in communications systems," *IEEE Trans. Commun.,* vol. COM-19, Oct. 1971.
[7] M. Cedervall and R. Johannesson, "A fast algorithm for computing distance spectrum of convolutional codes," *IEEE Trans. Inform. Theory,* vol. 35, no. 6, Nov. 1989.
[8] P. Thitimajshima, "Les codes convolutifs récursifs systématiques et leur application à la concaténation parallèle," (in French), Ph.D. no. 284, Université de Bretagne Occidentale, Brest, France, Dec. 1993.
[9] E. Dunscombe and F. C. Piper, "Optimal interleaving scheme for convolutional coding," *Electron. Lett.,* vol. 25, no. 22, pp. 1517–1518, Oct. 1989.
[10] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory,* vol. IT-20, pp. 248–287, Mar. 1974.
[11] J. Hagenauer, P. Robertson, and L. Papke, "Iterative ("TURBO") decoding of systematic convolutional codes with the MAP and SOVA algorithms," in *Proc. ITG'94,* 1994
[12] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *ICC'93,* Geneva, Switzerland, May 93, pp. 1064–1070.
[13] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near optimum decoding of products codes," in *GLOBECOM'94,* San Francisco, CA, Dec. 94, pp. 339–343.

Claude Berrou (M'84) was born in Penmarc'h, France in 1951. He received the electrical engineering degree from the "Institut National Polytechnique", Grenoble, France, in 1975.

In 1978, he joined the Ecole Nationale Supérieure des Télécommunications de Bretagne (France Télécom University), where he is currently a Professor of electronic engineering. His research focuses on joint algorithms and VLSI implementations for digital communications, especially error-correcting codes, and synchronization techniques.

Alain Glavieux was born in France in 1949. He received the engineering degree from the Ecole Nationale Supérieure des Télécommunications, Paris, France in 1978.

He joined the Ecole Nationale Supérieure des Télécommunications de Bretagne, Brest, France, in 1979, where he is currently Head of Department Signal and Communications. His research interests include channel coding, communications over fading channels, and digital modulation.