Treillis à complexité réduite pour le décodage de codes à longueur variable

Gholam-Reza MOHAMMAD-KHANI, Chang-Ming LEE, Michel KIEFFER, Pierre DUHAMEL

LSS — CNRS – Supélec – Université Paris-Sud
Plateau de Moulon - F-91192 Gif-sur-Yvette Cedex
Tél. : 00 33 1 69 85 17 32 – Fax : 00 33 1 69 85 17 65
{Gholam-Reza.Mohammad-Khani,lee,kieffer,Pierre.Duhamel}@lss.supelec.fr

Résumé – De nombreux algorithmes ont été proposés pour le décodage souple de données codées à l'aide de codes à longueur variable (CLV), la plupart travaillant avec des treillis. Pour un code réaliste, ces treillis sont très complexes à cause du nombre de mots de code à considérer. Cet article présente un algorithme de regroupement de mots de code à longueur variable en un nombre minimal de classes, ce qui permet de réduire significativement la complexité du décodage souple. Un exemple sur les CLV de la norme H263+ ainsi qu'une d'application à la localisation des frontières de blocs de texture H263+ sont présentés.

Abstract – Many trellis-based soft decoding techniques have been proposed for data encoded using variable-length codes (VLC). However, for actual VLC tables, these trellises are too complex to allow real-time soft decoding. This paper presents an algorithm for grouping VLC codewords into classes, which allows significant reductions of the size of the resulting trellises and of the complexity of soft decoding. Illustrations are provided on the VCL table used for texture encoding of H263+. The performances of decoding techniques for the localization of block frontiers in a bitstream generated by an H263+ coder are also presented.

1 Introduction

Dans de nombreux standards de compression, tel que JPEG, H263+ ou le mode de base de H264, une étape de codage à longueur variable (CLV) des données est effectuée. Les trains binaires générés par un CLV sont particulièrement sensibles à l'égard d'erreurs de transmission, principalement à cause de la perte de synchronisation qui peut résulter de certaines erreurs introduites par le canal.

Une solution classique consiste à insérer dans le train binaire des marques de synchronisation ou à faire appel à un codage canal puissant. Cependant, ceci résulte en une augmentation significative du débit binaire sur le canal. Des techniques plus récentes ont exploité la structure particulière des codes CLV pour développer des algorithmes de décodage souple de trains binaires issus d'un CLV [1–5]. L'ensemble des successions de mots de code possibles est décrite à l'aide de treillis exploitant certaines informations supposés disponibles *a priori* au niveau du décodeur (nombre de mots de code, longueur du train binaire à décoder, *etc.*). Les performances obtenues sont bien meilleures que celles atteintes par des décodeurs classiques. Des performances encore meilleures peuvent être obtenues en tenant compte de la syntaxe du train binaire généré par le codeur source considéré, voir par exemple [6, 7].

Toutes ces méthodes ont en commun la construction et l'élagage de treillis intégrant plus ou moins d'information *a priori*. La complexité de ces treillis peut devenir très importante lorsque des CLV comportant un nombre important de mots de code sont considérés. Cet article présente un algorithme de construction de tables simplifiées pour le décodage de données issues d'un CLV. Les mots de code sont regroupés en un nombre minimal de classes. Au lieu de travailler sur l'ensemble des mots de code, les algorithmes de décodage souple peuvent utiliser ces classes en nombre réduit, ce qui diminue la complexité des treillis et les temps de calcul.

Le paragraphe 2, présente brièvement un type de treillis utilisé pour le décodage de données issues d'un CLV. Le paragraphe 3 décrit le principe de l'algorithme de simplification des mots de code utilisés par un CLV donné. Un exemple de simplification des tables utilisées pour le codage de la texture par H263+ est présenté au paragraphe 5. Enfin, les performances d'un algorithme de décodage à entrées souples utilisant une table normale sont comparées à celle d'un algorithme utilisant une table compactée dans le paragraphe 4.

2 Décodage souple de CLV

On considère une source X générant des symboles appartenant à un alphabet $\mathcal{X} = \{X_1, \ldots, X_K\}$. A chaque symbole X_k est associé une probabilité d'occurence p_k. Un CLV associe à chaque symbole un mot de code $\mathbf{x}_k$ de $\ell_k = \ell(\mathbf{x}_k)$ bits, tel que $\ell_{\min} \leqslant \ell_k \leqslant \ell_{\max}$ pour tout $k \in \{1, \ldots, K\}$. A une suite de N symboles de la source peut donc être associé un vecteur de N mots de code résultant du CLV

$$\mathbf{x}_1^N = (\mathbf{x}(1), \ldots, \mathbf{x}(N)), \text{ avec } \sum_{i=1}^{N} \ell(\mathbf{x}(i)) = L.$$

Cette suite peut aussi être vu comme un vecteur de L bits $\mathbf{b}_1^L = (b_1, \ldots, b_L)$. Lorsque $\mathbf{b}_1^L$ est placé à l'entrée d'un canal de transmission, la sortie du canal est notée $\mathbf{y}_1^L = (y_1, \ldots, y_L)$.

Au niveau du décodeur, lorsque seuls L et la table des mots de code utilisée par le CLV sont connus, l'estimée $\hat{\mathbf{x}}$ au sens du maximum de vraisemblance de $\mathbf{x}_1^N$ est

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x} \in \mathcal{S}_L} p\left(\mathbf{y}_1^L \mid \mathbf{x}\right), \tag{1}$$

où $\mathcal{S}_L$ est l'ensemble de toutes les suites de mots de codes de L bits pouvant être générées par le CLV considéré. Lorsque le canal est gaussien sans mémoire, (1) se traduit par la minimisation sur $\mathcal{S}_L$ d'une norme euclidienne.

Pour résoudre ce problème de minimisation sous contrainte, [8] propose la construction d'un treillis représentant l'ensemble des suites de mots de code de longueur totale L pouvant être généré par un CLV. Une fois le treillis disponible, des algorithmes tels que SOVA [9] ou BCJR [10] peuvent être employés. Dans le treillis proposé, chaque nœud (a) represente la fin d'une suite de mots de code dont la longueur cumulée est a bits. Chaque branche correspond à un mot de code, ainsi, un groupe de branches parallèles reliant les nœuds (a) et $(a+\ell)$ representent tous les mots de code de ℓ-bits. La figure 1 montre qu'après un certain temps, la structure devient périodique. Chaque période du treillis contient au plus $\ell_{\max}$ nœuds. Le nombre total de nœuds est égal à L, le nombre de bits du vecteur $\mathbf{y}_1^L$, cependant, le nombre de branches arrivant et émergeant d'un nœud donné peut être très grand. Si l'on considère par exemple le CLV réalisé pour la texture dans la norme H263+, pour les mots de code de 10 bits, 42 branches parallèles relient $(a-10)$ et (a). Pour le décodage, le nombre de métriques de branche à évaluer est donc très important.

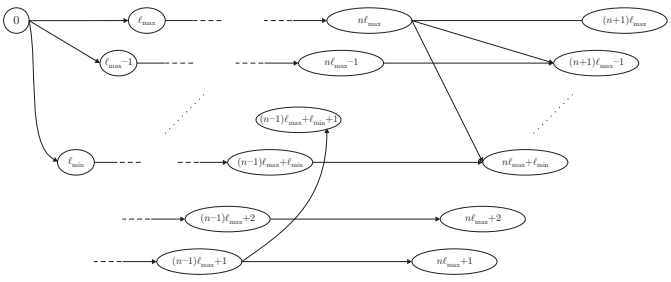

FIG. 1: *Structure de trellis pour des CLV de longueur comprise entre $\ell_{\min}$ et $\ell_{\max}$*

3 Simplification de tables de CLV

Une méthode permettant de réduire la complexité de décodage consiste à utiliser une table de mots de code de taille réduite. Cet article présente un algorithme de regroupement des mots de code de même longueur en classes. Ainsi, par exemple, les 16 mots de code de 8 bits utilisés pour le codage de la texture dans la norme H263+

$$\mathcal{A}_8 = \{00100000, 00100001 \ldots 00101110, 00101111\}$$

sont formés du préfixe 0010 et de tous les suffixes possible de 4 bits. $\mathcal{A}_8$ peut ainsi être décrit de manière compacte par 0010\$\$\$\$, où \$ représente soit 0, soit 1. A l'aide de cette simplification, dans le treillis présenté au paragraphe 2, 16 branches parallèles entre les nœuds (a) et $(a+8)$ peuvent être remplacées par une branche unique. Le principe de l'algorithme proposé généralise la technique de réduction présenté précédemment.

3.1 Notations et définitions

On considère l'ensemble des N mots de code de L bits, $\mathcal{A}_L = \{\mathbf{a}_1, \ldots, \mathbf{a}_N\}$, utilisé par un CLV donné. Le cardinal de $\mathcal{A}_L$ est noté $|\mathcal{A}_L|$. Il s'agit de trouver un regroupement des éléments de $\mathcal{A}_L$ de manière à obtenir un nombre minimal de représentants pour ces mots de code.

Soit $\mathbb{B} = \{0,1\}$, l'ensemble des mots binaires, qui peut être étendu en $\overline{\mathbb{B}} = \{0,1,\$\}$ avec l'élément *indéterminé,* correspondant à $\$ = \{0,1\}$. Une *classe* $\mathbf{c}$ est un élément de $\overline{\mathbb{B}}^L$. Soit $\mathcal{S}(\mathbf{c})$, l'ensemble des éléments de $\overline{\mathbb{B}}^L$ correspondant à $\mathbf{c}$. Ainsi, si $\mathbf{c} = (c_1, \ldots, c_L) \in \overline{\mathbb{B}}^L$, alors

$$\begin{aligned}\mathcal{S}(\mathbf{c}) \quad = \quad & \left\{(x_1, \ldots, x_L) \in \mathbb{B}^L \text{ tq } x_i = c_i \text{ si } c_i \neq \$ \right. \\ & \left. \text{et } x_i \in \{0,1\} \text{ si } c_i = \$\right\}.\end{aligned}$$

Une *classification* $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_M\}$ de $\mathcal{A}_L$ est un ensemble de classes tel que

$$\begin{cases} \forall \mathbf{x} \in \mathcal{A}_L, \exists i \text{ tel que } \mathbf{x} \in \mathcal{S}(\mathbf{c}_i), \\ \forall \mathbf{x} \in \mathcal{S}(\mathbf{c}_i),\ i = 1, \ldots, M, \mathbf{x} \in \mathcal{A}_L. \end{cases}$$

La méthode de simplification proposée consiste ainsi à représenter $\mathcal{A}_L$ à l'aide d'une *classification minimale* $\mathcal{C}$, c'est-à-dire une classification comportant un nombre minimal de classes.

Avant de présenter l'algorithme de construction de la classification minimale, quelques notions complémentaires doivent être introduites. La distance de Hamming $d(\cdot,\cdot)$ se généralise de $\mathbb{B}$ à $\overline{\mathbb{B}}$. Ainsi, si $(x,y) \in \overline{\mathbb{B}}$, $d(x,y) = 1$ si $x \neq y$ et $d(x,y) = 0$ sinon. La distance de Hamming entre deux classes est la somme des distances entre les composantes de ces classes. Deux classes sont *adjacentes* lorsque leur distance de Hamming est de un. L'*ordre* $o(\mathbf{c}) = \log_2(|\mathcal{S}(\mathbf{c})|)$ d'une classe $\mathcal{C}$ correspond au nombre d'éléments indéterminés de cette classe. Deux classes adjacentes $\mathbf{c}_1$ et $\mathbf{c}_2$ peuvent être *réunies* pour former une nouvelle classe $\mathbf{c} = \mathbf{c}_1 \sqcup \mathbf{c}_2$, on a alors $\mathcal{S}(\mathbf{c}) = \mathcal{S}(\mathbf{c}_1) \cup \mathcal{S}(\mathbf{c}_2)$ et $o(\mathbf{c}) = o(\mathbf{c}_1) + 1$.

3.2 Construction d'une classification minimale

Pour des raisons de place, seul le principe de l'algorithme de construction d'une classification minimale est présenté ici. Le détail de l'algorithme ainsi qu'une preuve d'optimalité est présenté dans [11].

L'algorithme a pour entrée un ensemble $\mathcal{A}_L$ de mots de codes de longueur L. Chaque élement de $\mathcal{A}_L$ est également une classe d'ordre 0. Une première classification $\mathcal{C}_0$ de $\mathcal{A}_L$ est formée à l'aide des éléments de $\mathcal{A}_L$. Un ensemble de classes $\mathcal{C}_1$ est formée de toutes les classes adjacentes de $\mathcal{C}_0$ qu'il est possible de réunir. $\mathcal{C}_1$ n'est plus nécessairement une classification de $\mathcal{A}_L$ (par exemple, s'il existe un élément $\mathbf{c} \in \mathcal{C}_0$ n'ayant pas de classe adjactente, alors $\mathbf{c} \notin \mathcal{C}_1$. La procédure est itérée jusqu'à l'obtension d'une classe $\mathcal{C}_{k_{\max}}$ ne comportant plus aucune classe adjacente.

L'union de tous les ensembles ainsi obtenus $\overline{\mathcal{C}} = \bigcup_{k=0}^{k_{\max}} \mathcal{C}_k$ est une classification *maximale* de $\mathcal{A}_L$, c'est-à-dire qu'elle contient toutes les classes qu'il est possible de former par réunion de classes adjacentes obtenues à partir des éléments de $\mathcal{A}_L$. Une classification minimale $\underline{\mathcal{C}}$ de $\mathcal{A}_L$ est alors un sous-ensemble de $\overline{\mathcal{C}}$ qui reste une classification de $\mathcal{A}_L$. L'obtension de $\underline{\mathcal{C}}$ se fait en deux étapes. La première consiste à éliminer de $\overline{\mathcal{C}}$ tous les éléments des ensembles intermédiares $\mathcal{C}_k$ qui ont pu être regroupés en classes appartenant à $\mathcal{C}_{k+1}$. La classification $\mathcal{C}$ est alors obtenue. Pour former la classification minimale $\underline{\mathcal{C}}$, une méthode combinatoire consistant à éliminer un maximum de

classes de $\mathcal{C}$ tout en vérifiant que le sous-ensemble obtenu reste une classification de $\mathcal{A}_L$ est ensuite considéré.

Cette technique peut fournir une classification minimale pour laquelle certains mots de code peuvent appartenir à plusieurs classes simultanément. Cette situation peut poser des problèmes lors du décodage à l'aide d'algorithmes de type MAP, la somme des probabilités *a priori* des classes n'étant plus égale à 1. Une solution à ce problème nécessite une adaptation de l'algorithme précédent, présentée dans [11].

Exemple 1 *Considérons l'ensemble $\mathcal{A}_7$ formé de 24 mots de code de 7 bits.*

$$\begin{aligned}\mathcal{A}_7 \quad = \quad & \{\mathbf{000}0000, \mathbf{000}0001, ..., \mathbf{000}1010, \mathbf{000}1011,\\ & \mathbf{0010}00, \mathbf{0010}01, \ldots, \mathbf{0010}11,\\ & \mathbf{0101}00, \mathbf{0101}01, \ldots, \mathbf{0101}11,\\ & \mathbf{1010}00, \mathbf{1010}01, \ldots, \mathbf{1010}11\}.\end{aligned}$$

Le calcul de la classification maximale et l'application de la procédure d'élimination des classes inutiles permet d'obtenir la classification suivante de $\mathcal{A}_7$

$$\mathcal{C}'_7 \quad = \quad \{0000\$\$\$, 000\$0\$\$, 00\$01\$\$, 0\$010\$\$, \$0101\$\$\}.$$

Une méthode combinatoire permet ensuite de fabriquer une classification minimale $\underline{\mathcal{C}_7} = \{0000\$\$\$, 0\$010\$\$, \$0101\$\$\}$ à partir de $\mathcal{C}'_7$.

4 Propriétés

Les tables de CLV obtenue par la procédure de simplification décrite au paragraphe 3 contiennent légèrement moins d'information que les tables initiales. Ceci est du au regroupement des mots de code en classes. Cependant, l'impact de cette perte d'information peut être très limité, selon l'algorithme de décodage utilisé.

Si un décodage au sens du maximum de vraisemblance est utilisé, l'algorithme utilisant les tables simplifiées fournira les mêmes résultats que s'il avait utilisé les tables complètes, en effet, les probabilités *a priori* des symboles ne sont pas utilisées.

Lorsque l'algorithme de décodage cherche à fournir une estimée au sens du maximum de vraisemblance et fournit des sorties souples (en utilisant par exemple SOVA [9], [12]), on peut montrer qu'à nouveau, les résultats sont identiques à ceux obtenus avec une table complète.

Pour un décodeur optimal utilisant un critère de maximum *a posteriori*, tel que celui présenté dans [13], le résultat restera optimal si les probabilités *a priori* de chacun des mots de code regroupés au sein d'une même classe sont les mêmes. Comme les mots de code qui peuvent être regroupés doivent avoir la même longueur, il est raisonnable de penser que cette condition sera vérifiée. Cependant, dans le cas d'une utilisation des tables simplifiées dans le cadre d'une procédure itérative, les différences entre les probabilités *a priori* des mots de code formant une classe pourront être source de sous-optimalité.

5 Application

Dans [7], un algorithme de décodage des informations de texture générées par un codeur H263+ est considéré. Chaque bloc de texture de 8×8 pixels subit un CLV puis est placé dans des paquets qui sont ensuite envoyés sur le canal. Un paquet contient en général plusieurs blocs de texture. Le décodeur proposé par [7] comprend deux étapes. La première consiste à localiser les blocs de texture dans un paquet. La seconde consiste à décoder chaque bloc ainsi localisé en exploitant certaines contraintes liées à la syntaxe du train binaire que peut générer un codeur H263+, voir [6] pour plus de détails.

TAB. 2: *Temps de décodage moyen (en ms/bloc)*

	ML, table CLV		MAP, table CLV	
	initiale	compacte	initiale	compacte
INTRA	165	62 (-62%)	242	92 (-62%)
INTER	356	108 (-70%)	445	131 (-71%)

Pour la première étape, seul le nombre de blocs que comporte un paquet est connu *a priori*. Pour localiser la position des blocs, le treillis présenté au paragraphe 2 doit être modifié pour faire apparaître comme propriété de chaque nœud le nombre de blocs accumulés. Ceci revient à ajouter une dimension au treillis. Pour chaque mot de code, il convient également de distinguer ceux qui marquent la fin d'un bloc.

Ainsi, les 204 mots de code dédiés au codage de la texture dans le standard H263+ sont partitionnés en deux sous-ensembles $\mathcal{A}$ et $\mathcal{E}$ contenant respectivement les mots de code ne marquant pas et marquant la fin d'un bloc. La table 1 rassemble les résultats obtenus par la procédure de simplification des tables de CLV appliquée aux mots de code de même longueur de $\mathcal{A}$ et de $\mathcal{E}$.

Les 204 mots de code peuvent être regroupés en 34 classes. Un algorithme de Viterbi est ensuite employé pour la localisation des frontières de blocs dans des paquets de texture. Les performances sont données en terme de taux d'erreur bloc pour des images codées INTER et INTRA sur la figure 2 et en terme de temps de décodage moyen par bloc dans le tableau 2.

Avec des temps de calcul presque trois fois moindres, les performances sont la plupart du temps très semblables, excepté pour le décodage au sens du maximum *a posteriori* des blocs de type INTRA, où une perte de l'ordre de 0.2 dB peut être notée. La diminution de performances dans le cas des paquets de type INTRA est due au fait que les mots de code regroupés dans une même classe n'ont pas toujours des probabilités *a priori* identiques. Le regroupement se fait alors avec une perte d'information. Dans le cas de paquets de type INTER, les probabilités *a priori* des mots de code regroupés sont plus proches, ce qui explique d'une part des performances du décodeur MAP à peine meilleures que celles du ML et d'autre part, une dégradation moins importante lors de l'utilisation de la table CLV simplifiée.

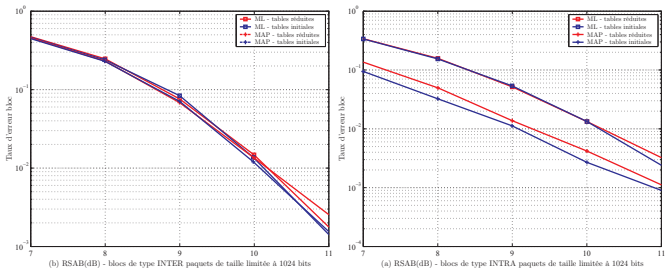

FIG. 2: *Décodage de paquets de blocs codés INTER et INTRA*

TAB. 1: *Classification minimale des mots de code utilisés pour le codage de la texture dans la norme H263+. Les mots marquant la fin d'un bloc (EOB) sont distingués des autres*

Longueur	EOB	Nb de mots de code	Classification minimale	Nb de classes
3	0	2	10$	1
4	0	2	110$	1
5	0	4	111$$	1
	1	2	0111$	1
⋮	⋮	⋮	⋮	⋮
10	0	24	00010010$$, 0001000$$$, 0000111$$$, 00001101$$	4
	1	18	00001100$$, 0000101$$$, 00001001$$, 000010001$	4
11	0	20	000010000$$, 0000001$$$$	2
	1	8	00000001$$$	1
12	0	12	0000000011$$, 000001000$$$	2
	1	12	0000000010$$, 000001001$$$	2
13	0	16	000001010$$$$	1
	1	16	000001011$$$$	1
Total		204		34

6 Conclusion

Cet article présente un algorithme de simplification des tables utilisés pour le décodage de CLV à l'aide d'algorithme de type maximum de vraisemblance ou maximum *a posteriori*. Certains mots de code peuvent être regroupés au sein de classes, ce qui permet de réduire la complexité des algorithmes de décodage. Une preuve d'optimalité de la procédure présentée sera fournie dans la version longue de l'article.

Dans l'exemple traité, la simplification a permi de représenter 204 mots de code par 34 classes, ce qui réduit largement la complexité des treillis représentatnt l'ensemble des succession de mots de code possibles pour une longueur de séquence codée reçue donnée. Les performances du décodeur au sens du maximum de vraisemblance restent inchangées lorsque des tables simplifiées lui sont fournies, mais le temps de calcul est presque trois fois moindre. Pour un décodeur utilisant un critère de maximum *a posteriori*, une légère perte d'optimalité peut être notée lorsque les probabilités *a priori* des mots de code regroupés au sein d'une même classe sont différents. Comme les mots de code qui peuvent être regroupés doivent avoir la même longueur, les différences restent faibles.

Références

[1] V. Buttigieg and P.G. Farrell. A MAP decoding algorithm for variable-length error-correcting codes. In *Codes and Cyphers: Cryptography and Coding IV*, pages 103–119, Essex, England, 1995. The Inst. of Mathematics and its Appl.

[2] V. B. Balakirsky. Joint source-channel coding with variable length codes. In *Proceedings of lEEE ISIT*, Ulm, Germany, 1997.

[3] N. Demir and K. Sayood. Joint source/channel coding for variable length codes. In *IEEE Data Compression Conf.*, pages 139–148, Snowbird, UT, 1998.

[4] K. Sayood, H. H. Otu, and N. Demir. Joint source/channel coding for variable length codes. *IEEE Trans. Commun.*, 48:787–794, 2000.

[5] R. Thobaben and J. Kliewer. On iterative source-channel decoding for variable-length encoded markov sources using a bit-level trellis. In *Proc. IV IEEE Signal Processing Workshop on Signal Processing Advances in Wireless Communications (SPAWC'03)*, Rome, 2003.

[6] H. Nguyen and P. Duhamel. Compressed image and video redundancy for joint source-channel decoding. In *Proc. Globecom*, 2003.

[7] C.M. Lee, M. Kieffer, and P. Duhamel. Soft decoding of vlc encoded data for robust transmission of packetized video. In *Proceedings of ICASSP*, pages 737–740, 2005.

[8] S. Kaiser and M. Bystrom. Soft decoding of variable-length codes. In *Proceedings of the IEEE International Conference on Communications*, volume 3, pages 1203–1207, New Orleans, 2000.

[9] J. Hagenauer and P. Hoeher. A viterbi algorithm with soft-decision outputs and its applications. In *Proc. Globecom*, pages 1680–1686, Dallas, TX, 1989.

[10] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. Info. Theory*, 20:284–287, 1974.

[11] G. R. Mohammad-Khani, M. Kieffer, and P. Duhamel. Simplification of vlc tables with application to ml and map decoding algorithms. *IEEE Transactions on Communications*, 2005. submitted.

[12] B. Vucetic and J. Yuan. *Turbo Codes - Principles and Applications*. Kluwer, Dordrecht, 2000.

[13] R. Bauer and J. Hagenauer. Turbo-FEC/VLC-decoding and its application to text compression. In *Proceedings of the Conference on Information Sciences and Systems (CISS'00)*, Princeton University, USA, 2000.