Titre: Title:	Analysis and Error Performances of Convolutional Doubly Orthogonal Codes with Non-Binary Alphabets
Auteur: Author:	Xuhua Shen
Date:	2014
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Shen, X. (2014). Analysis and Error Performances of Convolutional Doubly Orthogonal Codes with Non-Binary Alphabets [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1488/

UNIVERSITÉ DE MONTRÉAL

ANALYSIS AND ERROR PERFORMANCES OF CONVOLUTIONAL DOUBLY ORTHOGONAL CODES WITH NON-BINARY ALPHABETS

XUHUA SHEN
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE ÉLECTRIQUE)
AOÛT 2014

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

ANALYSIS AND ERROR PERFORMANCES OF CONVOLUTIONAL DOUBLY ORTHOGONAL CODES WITH NON-BINARY ALPHABETS

présentée par : SHEN Xuhua
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

M. FRIGON Jean-François, Ph.D., président
M. CARDINAL Christian, Ph.D., membre et directeur de recherche
M. HACCOUN David, Ph.D., membre et codirecteur de recherche
M. LABEAU Fabrice, Ph.D., membre
M. SZCZECINSKI Leszek, Ph.D., membre

DEDICATION

Dedicated to my parents.

ACKNOWLEDGEMENTS

It is my pleasure to thank those who made this thesis possible.

First and foremost I offer my sincerest gratitude to my academic supervisors, Prof. David Haccoun and Prof. Christian Cardinal, who have supported me throughout my study. With their experience in miscellaneous fields and the prosperous research progress, they guided me through the journey of the harsh yet appealing study process. I attribute the level of my Ph.D. degree to their encouragement and effort, to which I owe my deepest gratitude.

I am indebted to my colleague, Dr. Eric Roy, whose previous work and suggestion inspired me on the course of my research progress. I am also grateful to Mr. Wael Jaafar for his help in the French language of this thesis.

Last but not least, thanks go to the Mr. Jean-Sébastien Décarie and Mr. Jean Bouchard, without whose technical support the massive simulations in the project would never be possible.

RÉSUMÉ

Récemment, les codes convolutionnels auto-orthogonaux de Massey ont été adaptés au décodage efficace moderne. Plus spécifiquement, les caractéristiques et propriétés d'auto-orthogonalité de ce groupe de codes ont été étendues aux conditions de double-orthogonalité afin d'accommoder les algorithmes de décodage itératif modernes, donnant lieu aux codes convolutionnels doublement orthogonaux notés codes CDOs. Ainsi *l'instar de l'algorithme de propagation de croyance (en anglais, Belief Propagation, BP)*, le décodage itératif à seuil, développé à partir de l'algorithme de décodage à seuil de Massey, peut aussi être appliqué aux codes CDOs. Cet algorithme est particulièrement attrayant car il offre une complexité moins élevée que celle de l'algorithme de décodage à propagation de croyance. Les codes convolutionnels doublement orthogonaux peuvent être divisés en deux groupes : les codes CDOs non-récursifs utilisant des structures d'encodage à un seul registre à décalage, et les codes CDOs récursifs (en anglais Recursive CDO, notés RCDO) construits à partir de proto-graphes. À des rapports signal-à-bruit E_b/N_0 modérés, les codes non-récursifs CDO présentent des performances d'erreurs comparables à celles des autres technique courantes lorsqu'ils sont utilisés avec l'algorithme de décodage à seuil, présentant ainsi une alternative attrayante aux codes de contrôle de parité à faible densité (en Anglais Low-Density Parity-Check codes, notés LDPC). Par contre, les codes CDOs récursifs RCDO fournissent des performances d'erreur très élevées en utilisant le décodage BP, se rapprocha de la limite de Shannon. De plus, dans l'étude des codes LDPC, l'exploitation des corps finis $GF(q)$ avec $q > 2$ comme alphabets du code contribue à l'amélioration des performances avec l'algorithme de décodage BP. Ces derniers sont appelés alors les codes LDPC q-aires.

Inspiré du succès de l'application des alphabets dans un corps de Galois de q éléments $GF(q)$, dans les codes LDPC, nous portons dans cette thèse, notre attention aux codes CDO utilisant les corps $GF(q)$ finis, appelés CDO q-aires. Les codes CDO récursifs et non-récursifs binaires sont ainsi étendus à l'utilisation des corps finis $GF(q)$ avec $q > 2$. Leurs performances d'erreur ont été déterminées par simulation à l'ordinateur en utilisant les deux algorithmes de décodage itératif : à seuil et BP. Bien que l'algorithme de décodage à seuil souffre d'une perte de performance par rapport à l'algorithme BP, sa complexité de décodage est substantiellement réduite grâce à la rapide convergence au message estimé. On montre que les codes CDO q-aires fournissent des performances d'erreur supérieures à celles des codes binaires aussi bien dans le décodage itératif à seuil et dans le décodage BP. Cette supériorité en termes de taux d'erreur est plus prononcée à haut rapport signal-à-bruit E_b/N_0. Cependant ces avantages sont obtenus au prix d'une complexité plus élevée, complexité évaluée par le nombre des dif-

férentes opérations requises dans le processus de décodage. Afin de faciliter l'implémentation des codes CDO q-aires, nous avons examiné l'effet des alphabets quantifiés dans la procédure de décodage sur les performances d'erreur. Il a été démontré que le processus de décodage nécessite une quantification plus fine que dans le cas des codes binaires. Par contre, en fonction des détails de l'implémentation de l'algorithme de décodage, le décodeur à valeurs quantifiées permet d'obtenir des performances comparables à celle du décodeur en nombres réels (c.à,d. à point flottant) tout en utilisant un plus petit nombre de bits de mémoire. Dans nos simulations, il est montré que la sélection aléatoire des poids de connexion dans la construction des codes CDO q-aires est plus avantageuse que celle utilisant des poids identiques. Ceci est dû aux caractéristiques des graphes de Tanner et des algorithmes à passage des messages (en Anglais message passing). Les seuils de décodage des codes CDO non-récursifs sont calculés à partir de l'évolution de la densité de la probabilité du message avec l'approximation Gaussienne. Au lieu d'une relation monotone d'ordre q, il est démontré que les seuils calculés dépendent de différents paramètres du code (ex : taux de codage, dimension de l'alphabet, etc.). Enfin, se basant sur les travaux de Costello portant sur les codes convolutionnels binaires, des bornes supèrieures et inférieures sur les distances libres des codes CDO q-aires ont été développées lesquelles sont par la suite utilisées dans l'analyse des performances d'erreur des codes q-aires. L'analyse des performances d'erreur des codes convolutionnels q-aires en fonction de leurs propriétés de distance a conduit, en considérant un décodage à maximum de vraisemblance, à des approximations sur les probabilités d'erreurs des codes lorsqu'un décodage itératif est utilisé. Ces bornes se sont avérées utiles pour expliquer l'amélioration des performances d'erreur des codes CDO q-aires lorsque la valeur de q augmente.

ABSTRACT

Recently, the self orthogonal codes due to Massey were adapted in the realm of modern decoding techniques. Specifically, the self orthogonal characteristics of this set of codes are expanded to the doubly orthogonal conditions in order to accommodate the iterative decoding algorithms, giving birth to the convolutional doubly orthogonal (CDO) codes. In addition to the belief propagation (BP) algorithm, the CDO codes also lend themselves to the iterative threshold decoding, which has been developed from the threshold decoding algorithm raised by Massey, offering a lower-complexity alternative for the BP decoding algorithm. The convolutional doubly orthogonal codes are categorized into two subgroups: the non-recursive CDO codes featured by the shift-register structures without feedback, while the recursive CDO (RCDO) codes are constructed based on shift registers with feedback connections from the outputs. The non-recursive CDO codes demonstrate competitive error performances under the iterative threshold decoding algorithm in moderate E_b/N_0 region, providing another set of low-density parity-check convolutional (LDPCC) codes with outstanding error performances. On the other hand, the recursive CDO codes enjoy exceptional error performances under BP decoding, enjoying waterfall performances close to the Shannon limit. Additionally, in the study of the LDPC codes, the exploration of the finite fields $GF(q)$ with $q > 2$ as the code alphabets had proved to improve the error performances of the codes under the BP algorithm, giving rise to the q-ary LDPC codes.

Inspired by the success of the application of $GF(q)$ alphabets upon the LDPC codes, we focus our attention on the CDO codes with their alphabets generalized with the finite fields; particularly, we investigated the effects of this generalization on the error performances of the CDO codes and investigated their underlying causes.

In this thesis, both the recursive and non-recursive CDO codes are extended with the finite fields $GF(q)$ with $q > 2$, referred to as q-ary CDO codes. Their error performances are examined through simulations using both the iterative threshold decoding and the BP decoding algorithms. Whilst the threshold decoding algorithm suffers some performance loss as opposed to the BP algorithm, it phenomenally reduces the complexity in the decoding process mainly due to the fast convergence of the messages. The q-ary CDO codes demonstrated superior error performances as compared to their binary counterparts under both the iterative threshold decoding and the BP decoding algorithms, which is most pronounced in high E_b/N_0 region; however, these improvements have been accompanied by an increase in the decoding complexity, which is evaluated through the number of different operations needed in the decoding process. In order to facilitate the implementation of the q-ary CDO

codes, we examined the effect of quantized message alphabets in the decoding process on the error performances of the codes. It is demonstrated that the decoding process of the q-ary CDO codes requires finer quantization than that of the binary CDO codes; however, depending on the detailed implementation of the decoding algorithm, the quantized decoder still achieves comparable error performances compared to the floating point decoder with much smaller number of bits. In our simulations, the random selection of the connection weights in the q-ary CDO codes proved to be advantageous over the identical weight selection, which is explained through the features of the Tanner graphs and the message passing algorithms. The decoding thresholds of the non-recursive CDO codes are calculated using the density evolution with the Gaussian approximation. Rather than demonstrating a monotonic relation with the field order q, the calculated thresholds are shown to have a more complicated relation with various code parameters (e.g., coding rate, alphabet size, etc.). Finally, following Costello's work on binary convolutional codes, we have extended with q-ary alphabets, the upper and lower bounds on the free distances of convolutional codes which are used to analyze the error performances of q-ary codes. In analyzing the connection between the error performances of q-ary convolutional codes and their distance properties, we approximated the error probabilities of these codes under the iterative decoding with those under the maximum likelihood decoding; in which case the derived bounds proved useful for explaining the improvements in the error performances of q-ary CDO codes as q increases.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ACRONYMS AND ABBREVIATIONS

AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BCJR	Trellis Decoding Algorithm by Bahl, Cocke, Jelinek and Raviv
BP	Belief Propagation
BPSK	Binary Phase Shift Keying
CDO	Convolutional Doubly Orthogonal
EXIT	Extrinsic Information Transfer
FFT	Fast Fourier Transform
LDPC	Low-density Parity-check
MAP	Maximum a Posteriori Probability
p.d.f.	Probability Density Function
PEG	Progressive Edge Growth
RCDO	Recursive Convolutional Doubly Orthogonal
SCDO	Simplified Convolutional Doubly Orthogonal
SPC	Single Parity-Check

CHAPTER 1

Introduction

The recent error correction coding techniques have largely followed the track of factor-graph based decoding and construction strategies, giving birth to a variety of coding schemes, such as the low-density parity-check (LDPC) codes, the convolutional LDPC codes, the braided codes, and the codes defined upon the finite fields, etc. The primary focus of our work is on the q-ary convolutional doubly orthogonal (CDO) codes, including such aspects as their decoding algorithms, error performances, the decoding thresholds, and distance properties, etc. In this chapter, we briefly investigate the historical aspects of the CDO codes, the works carried out by our predecessors, the scope of our study, and present the overall organization of the thesis.

1.1 Background

The optimal decoding algorithm for error correcting codes under various channel conditions is the maximum *a posteriori* (MAP) algorithm (DeGroot, 2004). However, due to the complexity in implementing the MAP algorithm, long error correcting codes are usually decoded instead with the belief propagation (BP) algorithm (Kschischang *et al.*, 2001), which is best described by the message passing processes on the corresponding Tanner graphs (Tanner, 1981; Tanner *et al.*, 2001) of these codes. A Tanner graph consists of two separate categories of nodes, representing the code symbols and the constraints upon these symbols; between these two groups of nodes, the messages being processed in the BP algorithm *propagate* back and forth before the decoder comes to a decision on the code symbols.

The most famous coding scheme applying the BP decoding algorithm is the low-density parity-check (LDPC) codes proposed by Gallager in his thesis (Gallager, 1963), where he applied the probabilistic decoding algorithm on this set of codes. After more than thirty years in dormancy, the LDPC codes were revisited by Mackay (Mackay, 1999, 2003; C.Davey, 1999), with the decoding algorithm extended with the continuous message alphabets in an effort to adapt the codes to the additive white Gaussian noise (AWGN) channel. The BP decoding algorithm enacts the decoding of long LDPC codes, achieving exceptional error performances with moderate decoding complexity. Furthermore, unlike in the Turbo codes (Berrou *et al.*, 1993), the LDPC codes eliminate the need for an interleaver, enjoying much smaller decoding latency. In order to simplify the BP decoding process, the Log-domain decoding was propo-

sed (Fossorier *et al.*, 1999), which employs the *Log likelihood ratios* (LLRs) as the messages in the decoding process ; furthermore, (Chung *et al.*, 2001a) proposed a quantized decoder based on the Log-domain decoding algorithm in the investigation of the distributions of the messages. The merits of the BP decoding algorithm, however, is compromised by the short-length cycles in the Tanner graphs of the LDPC codes. Hence, one of the major concerns for the study on the LDPC codes has been on the optimization of the graph structure to ameliorate the breakdown of the independence assumption which results in the sub-optimality of the BP decoding algorithm (Djurdjevic *et al.*, 2003; Hu *et al.*, 2001; Kou *et al.*, 2001; Mackay, 1999; Pauluzzi and Beaulieu, 2000; Rlitsky *et al.*, 2002; Ryan, 2004) ; the essence which lies in reducing the number of short-length cycles in the corresponding Tanner graphs of the LDPC codes. Some of these works adopt the features of other codes (Djurdjevic *et al.*, 2003) and the algebraic properties of the finite geometry (Kou *et al.*, 2001) to construct Tanner graphs free of cycles of length 4 ; others are highly computational, such as the progressive edge growth (PEG) (Hu *et al.*, 2001), which establishes connections on a Tanner graph in a edge-by-edge manner to exclude connections resulting in short-length cycles. However, all these optimization methods require a pre-defined degree distribution pair for the two sets of nodes in the Tanner graphs. The most reputable work in optimizing the distribution pairs is the density evolution (Richardson and Urbanke, 2001), which extends the work in (Gallager, 1963) in calculating the error probability of the BP decoding algorithm under the AWGN channel. The worst channel parameters under which the LDPC codes are capable of achieving error-free transmission is defined as the *decoding thresholds* in (Richardson and Urbanke, 2001), which are applied in the comparison of various degree distribution pairs. The extrinsic information transfer (EXIT) chart (ten Brink, 1999) provides still another method in the optimization of the degree distribution pairs for the LDPC codes by visualizing the information exchange between the variable nodes and the constraint nodes in the Tanner graphs.

Following the prosperity of the LDPC codes, in (Jimenez Felstrom and Zigangirov, 1999), the authors apply the convolutional concept onto the low-density codes, leading to the set of LDPC convolutional (LDPCC) codes. This set of new codes had been shown to outperform the block codes with the same length in terms of bit error rates (BERs). Furthermore, the convolutional nature of the codes enables simpler implementation for both the encoding and the decoding processes ; particularly, the LDPCC codes employ a *pipelined* decoder based on the shift-register structure, greatly reducing the decoding latency as compared to the LDPC block codes. The LDPCC codes are divided into two categories : the time invariant codes, which are described by fixed encoding structures, and the time varying codes, in which case the encoding structure depends on the particular time slot. Following (Jimenez Felstrom and Zigangirov, 1999), various works had been conducted on the LDPCC codes (Cardinal

et al., 2008; Tanner *et al.*, 2004; Mitchell *et al.*, 2008; Tavares *et al.*, 2007; Feltstrom *et al.*, 2009), some of which induced notable sub-branches of these codes. For instance, in (Mitchell *et al.*, 2008), the LDPCC codes are constructed based on *protographs*, which are replicated and interconnected to form larger Tanner graphs ; the braided codes introduced in (Feltstrom *et al.*, 2009) serves as an example of the generalized LDPCC codes, which applies the concept of the generalized Tanner codes (Boutros *et al.*, 1999; Miladinovic and Fossorier, 2005) and employs the Hamming codes (Mackay, 2003) as component codes in their construction.

The developments on the convolutional codes can be somehow linked to the work of Massey (Massey, 1963), where the convolutional self orthogonal codes were proposed together with their threshold decoding algorithm. The threshold decoding algorithm takes advantage of the orthogonal properties of this set of codes and detects an error when the sum of the parity checks exceeds a fixed value (the threshold). The convolutional self orthogonal codes were also recently redeveloped into the convolutional doubly orthogonal (CDO) codes (Cardinal *et al.*, 2003) in order to adapt to the iterative decoding algorithms ; the CDO codes are categorized into two subgroups : the recursive CDO (RCDO) and the non-recursive CDO codes. The iterative threshold decoding algorithm was proposed specifically for this set of codes as an alternative for the BP decoding algorithm with lower complexity (He *et al.*, 2009). The doubly orthogonal feature of this set of codes guarantees the independence among the messages in the first two rounds of the decoding process, leading to their exceptional error performances. Efforts had also been poured into the analysis of the doubly orthogonal conditions as well as the search for CDO codes with small constraint lengths (Haccoun *et al.*, 2005; He and Haccoun, 2005; He *et al.*, 2009). Loosening the double orthogonal condition leads to a set of codes with much shorter constraint length, referred to as the simplified CDO codes (Cardinal *et al.*, 2009), demonstrating much smaller decoding latencies with slightly compromised error performances. It is also shown in (Roy *et al.*, 2010) that the recursive CDO (RCDO) codes can be constructed based on the protographs, leading to a subclass of the time invariant LDPC convolutional codes, achieving substantially better error performances as opposed to the LDPC block codes with comparable lengths.

For both block codes and convolutional codes, the distance property attracts vast attention in assessment of these codes (Lin and Costello, 2004). Although calculating the exact distance spectrum for the codes with long length is barely possible, the evaluation of upper and lower bounds of the distances proved to be a reasonable approximation for the purpose. Bounds on the minimum distances for LDPC block codes are provided in (Gallager, 1963), based on which an approximation on the error probability of the probabilistic decoding had been provided. For the convolutional codes, the upper and lower bounds on the minimum distances were provided by Massey in (Massey, 1963). However, later studies showed that the

free distance serves as a more appropriate performance measure for the convolutional codes, which is bounded later by Costello in (Costello, 1974).

Although the binary symbol alphabet has dominated the study on the block and convolutional codes, the extensions of the code symbol alphabets were also proposed in the evolution of error correcting codes ; however, they escaped major attention of researchers. (Singleton, 1963) proved the existence of short-length maximum-distance codes with non-binary alphabets ; and proposed the methods to construct these codes. Accompanying his invention of the LDPC codes, Gallager (Gallager, 1963) also demonstrated the possibility of extending the LDPC codes with alphabets over *rings*. The consistency of algebraic structures with assorted modulation schemes was investigated in (Slepian, 1968), producing the famous *Slepian signal sets* ; other researchers later extended the work in (Slepian, 1968), concentrating on the signal sets and the codes matched to *groups* (Ingemarsson, 1973; Loeliger, 1991; Forney, 1991). The recent awareness of the non-binary alphabets was due to the investigation of the LDPC codes over the *finite fields* of q (Davey and Markay, 1998) elements, referred to as the q-ary LDPC codes, leading to superior error performances as compared to their binary counterparts. The improvements in the error performances of this set of codes is attributed to the complex structures in their constraint nodes, which relate these codes closely with the generalized binary LDPC codes (Boutros *et al.*, 1999). However, decoding simplicity is sacrificed for the improvements in error performances for the q-ary codes under the BP decoding algorithm ; simplified versions of the BP algorithm (Wymeersch *et al.*, June 2004; Song and Cruz, 2003; Declercq and Fossorier, 2007; Savin, 2008) had been introduced to combat the problem. In the analysis of this set of codes, however, due to the large number of dimensions of the messages in the BP algorithm for the q-ary LDPC codes, the determination of the decoding thresholds through the density evolution technique is only implementable either with simplified message alphabets and channel conditions (Kurkoski *et al.*, 2007; Rathi and Urbanke, 2002) or with the Gaussian approximation (Li *et al.*, 2009).

1.2 Objective and Scope

In light of the superior error performances of the q-ary LDPC codes over the binary codes, we concentrate our effort at extending the convolutional doubly orthogonal (CDO) codes onto the finite-field alphabets and analyzing the error performances of this new set of codes under the iterative decoding algorithms.

The non-recursive CDO codes based on single-shift-register structures are generalized with their symbols defined over the finite field of order q, i.e., $GF(q)$. The iterative threshold decoding algorithm (Cardinal *et al.*, 2003) is extended to embrace the requirements of the q-ary

alphabets. The error performances of this set of codes under the iterative threshold decoding are then compared to those of the binary codes. For the fairness of comparison, we apply the binary phase shift keying (BPSK) modulation for all codes in our study ; the signals are then transmitted through a channel corrupted by additive white Gaussian noise (AWGN). The error performances of the q-ary non-recursive CDO codes under the BP decoding algorithm are also examined as opposed to the case under the iterative threshold decoding. Furthermore, for the BP decoding algorithm, the decoding thresholds of the q-ary non-recursive CDO codes are calculated using the density evolution with Gaussian approximation (Li *et al.*, 2009).

The recursive CDO codes based on the protograph design (Roy, 2006) are subsequently extended with the finite-field alphabets and decoded with the BP algorithm. This set of codes provide superior error performances as compared to the non-recursive codes. Furthermore, to facilitate the implementation of q-ary recursive CDO codes, the error performances of these codes under the quantized decoder are investigated.

The weights along the edges in the Tanner graphs of the q-ary CDO codes are randomly selected in our study. Both simulation results and analysis into the decoding algorithms demonstrate that the random selection is advantages over the identical selection of weights under the graph-based iterative decoding algorithms.

For Massey's self orthogonal codes (Massey, 1963), their minimum distances serves as an important indicator for the codes' error performances under the threshold decoding algorithm. However, for convolutional codes decoded with maximum likelihood (ML) algorithm and Viterbi decoding algorithm (Viterbi, 1967), the free distance is the prevailing parameter in approximating the *event error probabilities* of these codes. In this thesis, we aim to apply the distance properties of the q-ary convolutional codes in the analysis on error performances for our q-ary CDO codes. Similar to the case for the binary codes, it is not feasible to calculate the actual minimum distance or free distance for a given convolutional code whose shift registers are relatively long ; therefore, we have to resort to the upper and lower bounds on minimum distances and free distances for q-ary convolutional codes. Particularly, we aim to find the impact of the alphabet size q on the bounds on the free distances for q-ary convolutional codes, hence establish a link between the alphabet size and the error performances of our codes through their distance properties. Particularly, for the q-ary convolutional codes, upper and lower bounds on the free distances are derived in our study. Our bounds have their roots in previous works (Massey, 1963; Costello, 1974; Wozencraft and Reiffen, 1961). These bounds are then used in the estimation on the error probabilities for q-ary convolutional codes under the ML decoding algorithm. In order to derive the bounds on the free distances, we first propose an upper bound on the number of q non-zero sequences using the Chernoff bound (Wozencraft and Reiffen, 1961; Wozencraft and Jacobs, 1965) ; which is then applied

in evaluating the lower bounds for the minimum distance of q-ary convolutional codes. The bounds on the free distance are then derived as an extension on the bounds for the minimum distances.

It is worth noting that the estimated error probabilities do not close match the simulated BERs for your q-ary CDO codes due to several different reasons : i) our codes adopt the iterative decoding algorithms, e.g., BP algorithm and iterative threshold decoding algorithm, rather than the ML algorithm, the iterative decoding algorithms focuses on the decision on symbols while in ML decoding the decisions are made on code sequences ; ii) whilst the actual calculation for the error probabilities requires the knowledge of the entire distance spectrum (Lin and Costello, 2004), for codes with long shift registers, we may only estimate the free distances using their lower bound, which is only the power of the first term in the distance spectrum ; iii) the lower bound on the free distance addresses the existence of a 'good' q-ary convolutional code in terms of distances rather than provides a universal condition that any code should satisfy, hence a random code may have free distances lower than the given bound ; iv) in accordance to the requirements of iterative decoding algorithms based on graphs, our q-ary CDO codes focus on the doubly orthogonal conditions rather than on the distances in their construction. However, despite the above mentioned approximations, the bounds may still provide implications on the influences of q on error performances of the q-ary CDO codes, helping to explain the merits of moving onto finite fields of large order q. Although the benefits on the error performances of the q-ary linear codes when q increases had been attributed to their sparser Tanner graphs (Davey and Markay, 1998) ; we demonstrate that part of the improvements in BERs may also be linked to impact of q on the distance properties on these codes.

1.3 List of Contributions

The research conducted in this thesis were used to make the following contributions.

1. Extending the binary convolutional doubly orthogonal (CDO) codes with the q-ary alphabets.
2. Adapting the iterative decoding algorithms (the iterative threshold decoding and the BP decoding) for this new set of codes.
3. Investigation of the error performances of these codes under the additive white Gaussian noise (AWGN) channel.
4. Computation and comparison of the decoding complexities and decoding latencies of these new codes under the iterative decoding algorithms.
5. Calculation of the decoding thresholds of the q-ary CDO codes.

6. Investigating the effects of quantized message alphabets in decoding on the error performances the q-ary RCDO codes.
7. Investigating the upper and lower bounds on the distances of the q-ary linear codes.

1.4 Organization

Chapter 2 reviews the concepts of the binary convolutional doubly orthogonal codes and the low-density parity-check (LDPC) codes. The non-recursive CDO codes are defined with shift registers, while the recursive CDO codes most conveniently described with parity-check matrices. The doubly orthogonal conditions are provided for both categories of the codes, which guarantees the exceptional error performances of this set of codes under the iterative decoding algorithms. The bounds on the distances for the binary convolutional codes are also given as the background for our later discussion. The LDPC codes are defined with their parity-check matrices. The structures of the generalized LDPC codes are illustrated before we continue our discussion into the extension of the symbol alphabets with the finite fields. The error performances of these codes are then compared to those of the binary ones.

In Chapter 3, we extended the binary non-recursive CDO codes with symbol alphabets defined over the finite fields. The iterative threshold decoding algorithm and the BP algorithm are both adapted to meet the requirements of this set of new codes. The simulation results demonstrate that the q-ary non-recursive CDO codes are capable of achieving comparable error performances as the binary codes with much smaller decoding latencies. The decoding thresholds of the q-ary non-recursive CDO codes are examined through density evolution with the Gaussian approximation, where the calculated thresholds demonstrate dependencies on such factors as the code length, the field order, and the scaling parameter applied in the decoder.

In our study on the q-ary CDO codes, we assume randomly selected weights on the edges in the Tanner graphs. The performances of codes with random weights are compared to those with identical weights in Chapter 5. The random selection is shown to be advantageous over the identical selection regarding the error performances, which is explained through the analysis in both the Tanner graphs and the error probabilities in the BP decoding algorithm.

Following our work on the non-recursive codes, the generalization of the recursive codes are examined in Chapter 4. The q-ary recursive CDO codes are defined with their parity-check matrices and decoded with the belief propagation algorithm. Similar to the case with the non-recursive codes, we observe advantages on the error performances of q-ary recursive CDO codes as q gets large. Furthermore, the effects of quantized message alphabets in the decoding process on the error performances of these codes are investigated. It is demonstrated with

simulations that decoder with proper quantization achieves comparable error performances with the floating point decoder.

Finally, we present the bounds on the free distances of the q-ary convolutional codes in Appendix A. These bounds are derived based on the works of Gallger (Gallager, 1963) and Costello (Costello, 1974). The lower bound on the free distances of the q-ary convolutional codes is derived based on the ensembles of the codes while the upper bound applies to any single q-ary convolutional code. It is shown that these bounds are functions of such parameters as the field order, code rate, and the memory order of the codes. These bounds help in the explanation of the differences in the error performances of the CDO codes with different q values.

Chapter 6 briefly summarizes the work performed in this thesis and discusses potential future works.

CHAPTER 2

Literature Review

Our study concentrates on the area of the iteratively decoded error correcting codes based on the Tanner graphs (Tanner, 1981).

Stemming from Massey's work on the convolutional self orthogonal codes (Massey, 1963), the novel convolutional doubly orthogonal (CDO) codes constitute a set of sparse-graph codes. The doubly orthogonal conditions guarantee the independence among the messages in the iterative decoding for the first few rounds of iterations, leading to the outstanding error performances for this set of codes. Most importantly, the CDO codes enjoy simple encoding due to their shift-register-based encoders. The latency in the decoding process of this set of codes is proportional to the memory length of their shift registers; therefore, efforts had been poured into the search for the CDO codes with the smallest memory order. On the other hand, the low-density parity-check (LDPC) codes serve as the prototype of various modern coding techniques based on the Tanner graphs. Originally introduced by Gallager (Gallager, 1963), the study on the LDPC codes was extended by various researchers to cover such areas as the decoding algorithms under the AWGN channel (C.Davey, 1999), graph construction methods (Hu *et al.*, 2001; Fossorier, 2004), and decoding thresholds (Richardson and Urbanke, 2001; ten Brink, 1999), etc. Variants of the LDPC codes had been developed based on the local decoding concepts, the most famed of which are the generalized LDPC codes (Lentmaier and Zigangirov, 1999; Miladinovic and Fossorier, 2005; Boutros *et al.*, 1999), applying simple short-length codes as the constraint nodes in their Tanner graphs. The non-binary alphabets for the LDPC codes had also been mentioned by Gallager (Gallager, 1963), with integer rings directly applied on the LDPC codes; later versions of the non-binary LDPC codes include the LDPC codes defined over the finite fields (Davey and Markay, 1998), and the generalized groups (Rathi and Urbanke, 2002), etc.

This chapter summarizes the definitions and properties for various categories of the CDO codes and the LDPC codes. The iterative threshold decoding and the belief propagation (BP) decoding algorithms for these codes are described. Code construction schemes in the literature are also briefly investigated.

2.1 The Convolutional doubly orthogonal codes

2.1.1 Definition and encoding of CDO codes

Depending on their encoding structure, the convolutional doubly orthogonal codes are generally categorized into two subgroups : the non-recursive codes and the recursive codes. As their names indicate, previously transmitted symbols exert no impact on the encoding process of the non-recursive codes while that of the recursive codes is affected by the earlier outputs. In terms of their encoding diagrams, the recursive codes contain feedback connections from outputs onto the shift registers ; which is not shared by the non-recursive ones. In this section, the definitions are provided for the two groups of codes. For the purpose of compactness, we only discuss the codes within the sub-categories most relevant to our work, i.e., the non-recursive codes with a single shift register and the recursive codes defined with protographs.

Non-recursive CDO codes The non-recursive CDO codes are defined with shift-register structures. Although the number of shift registers in an encoder is arbitrary according to the requirements of the application, we only consider the non-recursive codes with a single shift register. The definition of codes with multiple shift registers can be found in (Cardinal *et al.*, 2003).

The encoder of a rate 1/2 non-recursive CDO code with single-shift-register structure is depicted in Fig. 2.1, where u_t is the information bit at time t and p_t denotes the corresponding parity bit. p_t is calculated as the *modulo* 2 sum of J connections to the shift register :

$$p_t = \sum_{i=1}^{J} \oplus u_{t-\alpha_i}, \tag{2.1}$$

where $\sum \oplus$ represents the mod 2 summation.

Therefore, the code is fully defined by the set of connection positions from the shift register

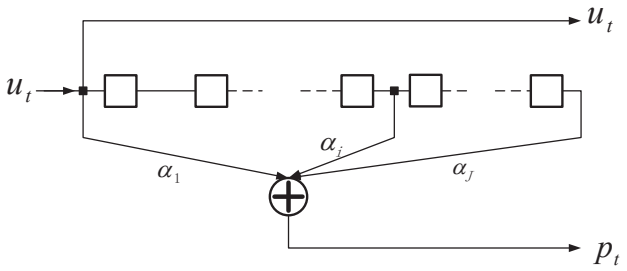


Figure 2.1 Encoding of non-recursive convolutional doubly orthogonal codes with single-shift-register structure.

$\{\alpha_i\}, i = 1, 2, \ldots, J$. For a code with the encoder depicted in Fig. 2.1 to be a convolutional doubly orthogonal code, the following conditions need to be satisfied :

1. the differences $(\alpha_i - \alpha_j)$, with $i \neq j$, are distinct ;
2. the differences of differences $(\alpha_i - \alpha_j) - (\alpha_l - \alpha_n)$ are distinct for all (i, j, l, n), $i \neq j$, $l \neq n$, $i \neq l$, $j \neq n$ except for the unavoidable repetitions ;
3. the differences of differences are distinct from the differences.

In (Cardinal *et al.*, 2009), a group of codes with condition 2 partially fulfilled is defined to be the *simplified* convolutional doubly orthogonal (SCDO) codes, enjoying much smaller memory order as compared to the CDO codes.

It is worth noting that the doubly orthogonal conditions for the codes with a single shift register was originally defined as *wide sense* in (Cardinal *et al.*, 2003), for condition 2 does not excludes the repetition among second order differences ; which is inevitable with the single-shift-register structure. However, when using CDO codes in the *strict sense*, this imperfection is overcome (Cardinal *et al.*, 2003).

Recursive CDO codes At time t, a rate b/c RCDO code receives b information symbols $\mathbf{u}_t = \{u_t^0, u_t^1, \ldots, u_t^{b-1}\}$ as its inputs and outputs c code symbols, represented by the vector $\mathbf{v}_t = \{v_t^0, v_t^1, \ldots, v_t^{c-1}\}$, where $u_t^i, v_t^i \in \{0, 1\}$. We only consider systematic codes in our study, i.e., $v_t^i = u_t^i$, for $i = 0, 1, \ldots, (b-1)$. In the output vector, the $(c-b)$ parity symbols are calculated with the parity-check equations :

$$v_t^i = \sum_{j=0}^{b-1} v_{t-\alpha_{j,i}}^j + \sum_{\substack{j=b \\ j \neq i-b}}^{c-1} v_{t-\alpha_{j,i}}^j, \tag{2.2}$$

for $i = b, b+1, \ldots, c-1$. Note that the summations are modulo 2-based.

The encoder of the q-ary RCDO codes is depiced in Fig. 2.2, which is realized with the observer canonical form (Lin and Costello, 2004). Each of the parity symbols is assigned a shift register, storing appropriately delayed versions of the information symbols and the parity symbols from the feedback ; the ith symbol is connected to the $\alpha_{i,j}^{th}$ delay cell of the j^{th} shift register.

Although the recursive CDO codes are most directly defined with shift-register diagrams, a more recent construction of this set of codes is based on the protograph approach (Roy *et al.*, 2010), where the author employs a parity-check matrix in specifying the CDO codes.

The RCDO binary semi-infinite parity-check matrix is represented using its D-transform

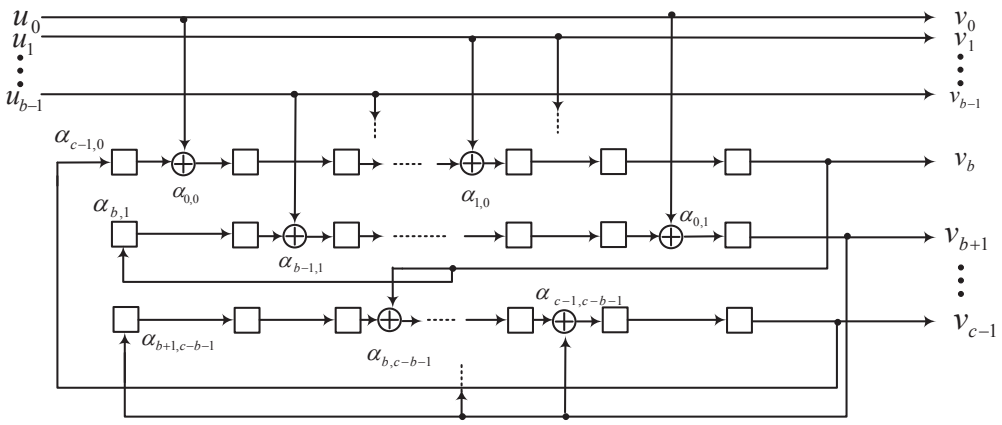


Figure 2.2 Encoding of rate b/c recursive convolutional doubly orthogonal codes.

polynomial representation :

$$\mathbf{H}^T(D) = \begin{bmatrix} h_{0,0}D^{\alpha_{0,0}} & \dots & h_{0,c-b-1}D^{\alpha_{0,c-b-1}} \\ h_{1,0}D^{\alpha_{1,0}} & \dots & h_{1,c-b-1}D^{\alpha_{1,c-b-1}} \\ \vdots & & \vdots \\ h_{c-1,0}D^{\alpha_{c-1,0}} & \dots & h_{c-1,c-b-1}D^{\alpha_{c-1,c-b-1}} \end{bmatrix}, \tag{2.3}$$

where D denotes the delay operator, and $\alpha_{i,j}$ represents the number of time slots that a symbol is delayed with respect to the initial symbol in the code sequence. Furthermore, $h_{i,j} \in \{0, 1\}$ indicates whether a connection exists between the output symbol and a shift register, i.e., if $h_{i,j} = 1$, the $i^{\,th}$ output symbols is connected to the $j^{\,th}$ shift register on the $\alpha_{i,j}^{\,th}$ delay cell.

The protograph (Mitchell *et al.*, 2008) representation of the RCDO codes is obtained from the $\mathbf{H}^T(D)$. We define the matrix $\mathbf{H}^T$ as

$$\mathbf{H}^T = \begin{bmatrix} h_{0,0} & h_{0,1} & \dots & h_{0,c-b-1} \\ h_{1,0} & h_{1,1} & \dots & h_{1,c-b-1} \\ \vdots & \vdots & & \vdots \\ h_{c-1,0} & h_{c-1,1} & \dots & h_{c-1,c-b-1} \end{bmatrix}. \tag{2.4}$$

A protograph is a bipartite graph defined by two sets of nodes and their connections. Each *variable node* in the protograph represents a row of $\mathbf{H}^T$ and each *constraint node* represents a column. Hence, the total number of the variable nodes and the constraint nodes in the protograph of a RCDO encoder is given by c and $(c - b)$ respectively. Fig. 2.3 is an example of a protograph with 6 variable nodes and 3 constraint nodes corresponding to the following

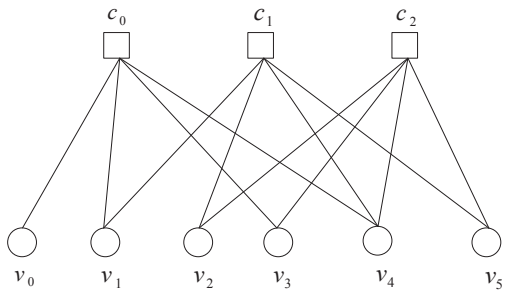


Figure 2.3 A protograph example with 6 variable nodes and 3 constraint nodes.

parity-check matrix :

$$\mathbf{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \tag{2.5}$$

The complete Tanner graph of a RCDO code can be obtained with two steps : i) duplicating the protograph for a certain number of times, ii) permuting the edges between all the duplicated protographs. For RCDO codes, the permutation of the edges is performed in accordance to the doubly orthogonal conditions, which are described as follows :

1. The differences $(\alpha_{k,n} - \alpha_{s,n})$ must be distinct from the differences $(\alpha_{k,m} - \alpha_{s,m})$, for $k \neq s, m \neq n$;
2. The differences $(\alpha_{k,l} - \alpha_{t,l})$ must be distinct from the difference of differences $(\alpha_{k,n} - \alpha_{s,n}) - (\alpha_{t,f} - \alpha_{s,f})$, for $k \neq t, k \neq s, s \neq t, f \neq n, l \neq n$;
3. The difference of differences $(\alpha_{k,n} - \alpha_{s,n}) - (\alpha_{p,f} - \alpha_{s,f})$ must be distinct from the difference of differences $(\alpha_{k,m}\alpha_{r,m}) - (\alpha_{p,g} - \alpha_{r,g})$, for $k \neq s, p \neq s, k \neq r, f \neq n, p \neq r, g \neq m$.

2.1.2 Iterative decoding algorithms based on graphs

The decoding of the CDO codes follows the message passing process on their corresponding Tanner graphs. As indicated previously, a Tanner graph is a bipartite graph consisting of two types of nodes : each code symbol in a CDO code is assigned a variable node and each parity-check equation is represented by a constraint node. The connections between the two types of nodes correspond to the entries in the parity-check matrix of the code. The number of

constraint nodes connected to a variable node is defined as the *degree* of that variable node; similarly, the degree of a constraint node equals the number of variable nodes incident to it.

Two types of *messages* are communicated back and forth between the two types of nodes in the decoding process. The message from a variable node to a constraint node is essentially the discrete probability density function (p.d.f.) indexed by possible values of the symbol, i.e., $\{0, 1\}$; and the message from a constraint node to a variable node concerns the conditional probabilities that its corresponding parity-check equation is satisfied given a fixed value of that variable node. The two types of messages are transmitted and updated at each node using the messages received by that node. The process continues for a predetermined number of iterations before the messages in the graphs are employed to determine the value of the code symbols. Although the iterative decoding algorithms are transformed into assorted versions (with their messages interpreted in various domains), the essence of the decoding remains unchanged as stated above.

The decoding of the CDO codes is performed either the BP decoding algorithm or the iterative threshold decoding algorithm. The two algorithms share a few common steps yet with different variable node processing; leading to the much smaller decoding complexity for the iterative threshold decoding than the BP decoding algorithm (He *et al.*, 2009). In Chapter 3, the error performances of the CDO codes are shown to be substantially affected by the choice of the decoding algorithm.

In the following, the main processing of iterative decoding algorithms for binary CDO codes are described.

The BP and the iterative threshold decoding algorithms During the first half of a single iteration, each constraint node receives the messages from all of its incident variable nodes and calculates for each of them a new message in return; these messages are then employed in the second half of the iteration by the variables in the computation for the messages to be transmitted back to their incident constraint nodes during the next iteration.

Updating for a constraint node The message computation at the constraint nodes is identical for the BP and the iterative threshold decoding algorithm. Consider a constraint node with degree d_c, i.e., it is connected to d_c variable nodes as depicted in Fig. 2.4. The constraint node c receives messages from its neighboring variable nodes $v_i, i = 1, 2, \ldots, d_c$. Let s_i denote the message from v_i. We assume the messages are Log likelihood ratios (LLRs), which are calculated using :

$$s_i = \log\left(\frac{\text{Prob}(v_i = 0)}{\text{Prob}(v_i = 1)}\right), \tag{2.6}$$

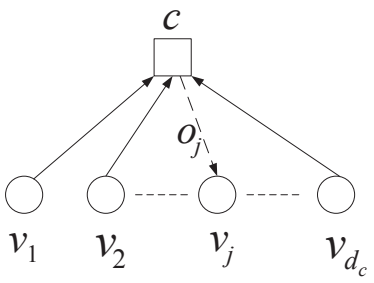


Figure 2.4 Message flow in constraint node updating for a degree-d_c constraint node.

for $i = 1, 2, \ldots, d_c$. The message o_j to be returned to the j^{th} variable node is computed using all except the j^{th} messages from the variable nodes (Ryan, 2004) :

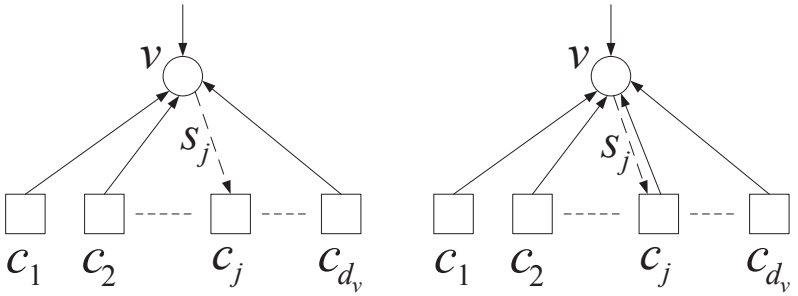
$$o_j = \prod_{i=1, i\neq j}^{d_c} \operatorname{sgn}(s_i) \cdot \phi\left(\prod_{i=1, i\neq j}^{d_c} \phi(|s_i|) \right), \tag{2.7}$$

where

$$\phi(x) = -\log[\tanh(\frac{x}{2})] = \log\left(\frac{e^x + 1}{e^x - 1}\right). \tag{2.8}$$

Note that in order to calculate the message for v_j, the message from v_j are *excluded* from the computation.

Furthermore, in the iterative threshold decoding, a constraint node receives messages in their *most updated* versions (Cardinal *et al.*, 2003), while that in the BP decoding only employs messages from the previous round of iteration ; the details of which will be explained in our discussion on the q-ary CDO codes in Chapter 3.



(a) BP decoding ; (b) Iterative threshold decoding ;

Figure 2.5 Message flow in variable node updating for a degree-d_v variable node.

Updating for a variable node The BP decoding algorithm and the iterative threshold decoding algorithm differ in the updating process at the variable nodes. As illustrated in Fig. 2.5, under the BP algorithm, the variable node excludes the message from a constraint node when computing the message to be transmitted back to it ; while under the iterative threshold decoding it employs *all* the incoming messages from the incident constraint nodes. Furthermore, an additional message from the channel is included in the variable node updating process for both the algorithms. In Fig. 2.5, a variable node v is connected to d_v constraint nodes. Denote o_i as the message from the constraint node c_i for $i = 1, 2, \ldots, d_v$, and let t denote the message from the channel, the updated message s_j for the constraint node c_j under the BP decoding algorithm takes the form (Ryan, 2004)

$$s_j = t + \sum_{i=1, i\neq j}^{d_v} o_i, \tag{2.9}$$

while that for the iterative threshold decoding is calculated using (Cardinal *et al.*, 2003)

$$s_j = t + \sum_{i=1}^{d_v} o_i. \tag{2.10}$$

2.1.3 Search and span minimization of CDO codes

The doubly orthogonal conditions, either for the recursive codes or the non-recursive code, can always be satisfied if no constraint is exerted upon the memory order (maximum length of the shift registers). However, the decoding latency of the CDO codes is proportional to the length of the shift registers and the number of iterations ; hence it is always desirable to construct CDO codes with the smallest possible memory order. Various search methods were proposed in the literature to fulfill the task (Haccoun *et al.*, 2005; Cardinal *et al.*, 2008, 2009; Roy *et al.*, 2010).

Heuristic search has been the primary tool in constructing good CDO codes. As an example, the search for non-recursive codes with a single shift register is described as follows (Haccoun *et al.*, 2005) :

For fixed value of J :

1. Initially, the number of connections J_0 is set at 0 ;
2. Add an element among natural integers arranged in ascending order ;
3. Test whether the newly formed $(J_0 + 1)$-dimensional vector satisfies the double orthogonal conditions, if it doesn't, go back to Step 2 ;

4. Run a random test to determine whether to keep the most recently found integer, if yes, J_0 is increased by 1, if not, go back to Step 2 ;
5. Step 2 to 4 are repeated until $J_0 = J$.

Since any addition or multiplication applied on a found connection position vector does not affect its validity, it is preferable to apply span (memory order) reduction techniques following the search method. There is, however, limit on the extent to which we may shorten the memory order, a lower bound has been given in (Haccoun *et al.*, 2005) :

$$\alpha_J \geq \frac{J^4 - 2J^3 + 7J^2 - 6J}{16}. \tag{2.11}$$

Fast parallel searching algorithms for the CDO codes with shortest spans were proposed recently (Kowarzyk *et al.*, 2008, 2012, 2013).

It is worth noting that it is preferable, under some scenarios, to relax the doubly orthogonal conditions in order to construct codes with relatively small memory order, leading to the *simplified* CDO (SCDO) codes as defined previously. The SCDO codes enjoy relatively small memory order and accommodate code rate other than 1/2 (Cardinal *et al.*, 2009).

2.1.4 Error performance of CDO codes

This section briefly presents the error performances of the typical rate 1/2 non-recursive and recursive CDO codes under the additive white Gaussian noise (AWGN) channel obtained through computer simulation, as illustrated in Fig. 2.6 (Haccoun *et al.*, 2005; Cardinal *et al.*, 2008, 2009). The typical values for the memory lengths of non-recursive and recursive CDO codes are listed in Table 2.1 (Haccoun *et al.*, 2005) and Table 2.2 (Roy, 2011).

Table 2.1 Memory orders of typical non-recursive CDO codes (Haccoun *et al.*, 2005).

J	5	8	10	12	15	18	21	23
α_J	33	459	1698	5173	23193	75629	212456	391403

Table 2.2 Memory orders of typical recursive CDO codes (Roy, 2011).

b/c	4/8	5/10	6/12	7/14	8/16
$\alpha_{\max}$	33	33	34	34	62

The non-recursive codes employs the single-shift-register structure, with $J = 15$ connections from the shift register to the parity symbol, decoded with the iterative threshold decoding algorithm. First note that only slight improvement in its error performances is observed

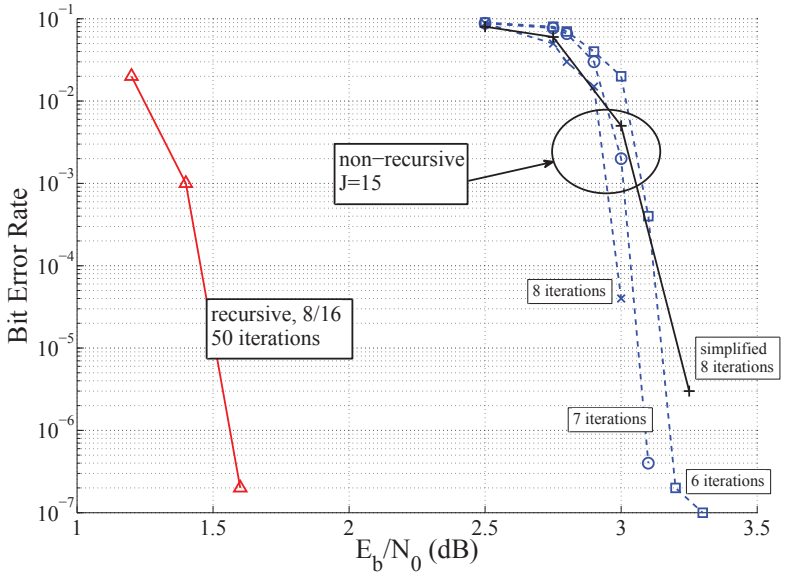


Figure 2.6 Error performances of rate 1/2 convolutional doubly orthogonal codes (Haccoun *et al.*, 2005; Cardinal *et al.*, 2008, 2009).

when the iteration number increases from 6 to 8, i.e., it is quite sufficient to apply 8 iterations for this code. The error performances of a simplified CDO code with $J = 15$ are plotted as a comparison. After the 8 th iteration, the SCDO code suffers a performance degradation of 0.2 dB in high E_b/N_0 region ; but it performs comparably with the CDO code in low E_b/N_0 region. This degradation is a slight cost for the improvement in decoding latency, i.e., the SCDO code has a memory order of around 3000, whilst that of CDO code reaches more than 20000.

The error BERs of the recursive CDO code presented in Fig. 2.6 is obtained using the BP algorithm. It uses the protograph-based design as discussed previously, with memory order 62, and therefore has a latency proportional to $63 \times 16 = 1008$. The RCDO code is observed to improve the error performance by approximately 1.6 dB as compared to the $J = 15$ non-recursive code. The improvement comes from both its recursive structure and the BP decoding algorithm. However, it is worth noting that for the BP algorithm to achieve good error performances, 50 iterations are required in the decoding process as compared to the 8 iterations in the iterative threshold decoding, greatly increasing the decoding latency (He *et al.*, 2009).

In addition to the bit error rates obtained through the computer simulations, another important performance indicator for the convolutional codes is their distance property, which is briefly reviewed in the following.

2.1.5 Distance properties of convolutional codes

The distance property is an important parameter for predicting the performances of linear codes. The *distance* between the two code sequences is defined to be the number of positions in which they differ. The *minimum distance* of a block code is the smallest distance between any two of the codewords for this block code. The distances of the convolutional codes, however, follows different definitions.

As mentioned previously, the *memory order* m of a rate $R = b/c$ convolutional code is the maximum length of the shift registers in its encoder. Once initiated, the encoder of a convolutional code continuously generates c code symbols during each time slot. In (Massey, 1963), an *initial codeword* is defined to be the first $n_A = (m+1)c$ output symbols from the encoder generated using the first $(m+1)b$ information symbols.

Definition 1 *The minimum distance d_{min} of a convolutional codes is defined to be the smallest number of symbols for which two initial codewords differ that do not have the identical sets of the first $(m+1)b$ information symbols.*

Later studies demonstrated that for the convolutional codes, a more appropriate measure on their distance properties is the free distance d_{free}, which is the smallest distance among all non-zero codewords starting from and eventually ending in an initial state, i.e., when the shift registers are filled with all-zero symbols. Obviously, $d_{\text{free}} \geq d_{\min}$.

As an example, consider the convolutional code with the encoding diagram as in Fig. 2.7 (Lin and Costello, 2004) with $m = 3$, its state transition diagram is given in Fig. 2.8. It can be verified from Fig. 2.8 that the minimum distance for this code $d_{\min} = 2$ with state transition $S_0S_0S_0S_1$; however its free distance $d_{\min} = 6$ with state transition $S_0S_1S_3S_6S_4S_0$.

Massey provided the upper and lower bounds on the minimum distances for convolutional codes (Massey, 1963) :

- There exists at least one binary convolutional code with code rate $R = b/c$ and constraint length n_A with minimum distance $d_{\min} \geq d$, where d is the largest integer satisfying
$$H(\frac{d}{n_A}) \leq 1 - R, \tag{2.12}$$
where $H(\lambda) = -\lambda \log \lambda - (1-\lambda)\log(1-\lambda)$ denotes the binary entropy function ;
- d_{free} of a binary convolutional code with code rate $R = b/c$ satisfies
$$d_{\text{free}} \leq \frac{1}{2}(n_A + c). \tag{2.13}$$

The bounds on d_{free} had been provided by Costello in (Costello, 1974) :

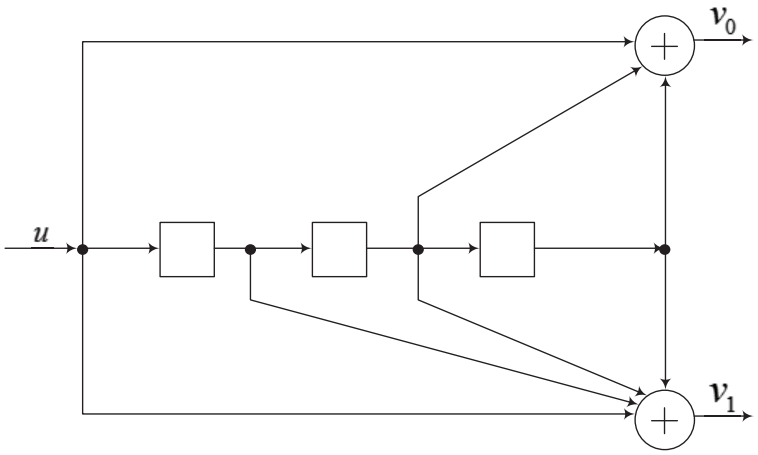


Figure 2.7 Example of the encoder of a convolutional code (Lin and Costello, 2004).

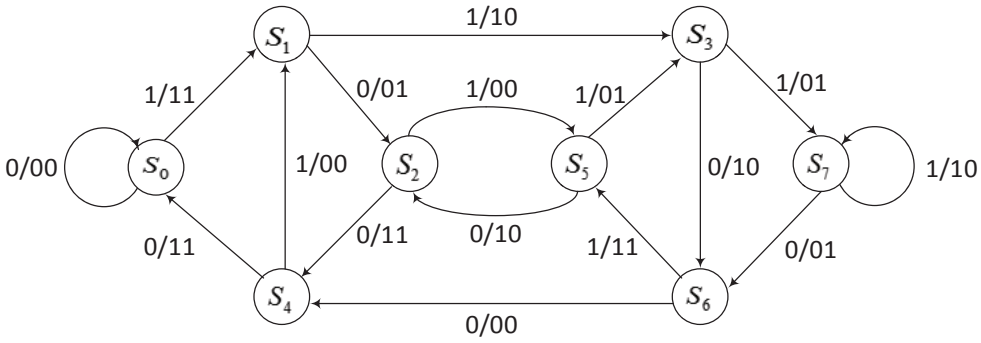


Figure 2.8 Example of the state transition diagram of a convolutional code (Lin and Costello, 2004).

– For code rate $R = b/c$, there exists a convolutional code such that

$$\lim_{m\to\infty} \frac{d_{\text{free}}}{n_A} \geq \begin{cases} 2H^{-1}(1-R), & \frac{3}{8} \leq R \leq 1; \\ \frac{2R(1-2^{2R-1})}{H(2^{2R-1})+2R-1}, & 0 \leq R \leq \frac{3}{8}; \end{cases} \tag{2.14}$$

– For any time invariant convolutional code,

$$d_{\text{free}} < \frac{n_A}{2} + \frac{1}{2R}\log n_A + \frac{1}{2}, \tag{2.15}$$

if

$$\frac{n_A}{b + \log n_A} > \frac{1}{R}. \tag{2.16}$$

2.2 LDPC codes and the non-binary alphabets

Prior to the CDO codes, the low-density parity-check (LDPC) codes attracted much attention as the prototype of a group of modern coding schemes depicted with graphs, boosting the development of other novel error correcting codes. This section briefly reviews the concepts and the methodologies used in the study of the LDPC codes, which inspires our study described in latter parts of the thesis.

2.2.1 Definition of the LDPC codes

The LDPC codes are most conveniently defined by their parity-check matrices; while the Tanner graph representation serves to facilitate the discussion on the decoding algorithms.

Matrix definition The codewords $\{\mathbf{c}\}$ of a rate K/N LDPC code form a vector space of N-dimensional binary vectors. An LDPC code is a K-dimensional subspace $\mathcal{C} = \{\mathbf{c}\}$ of this vector space satisfying a set of parity-check equations :

$$\mathbf{c}\mathbb{h}_i^T = 0 \quad \text{for} \quad i = 1, 2, \ldots, N-K; \tag{2.17}$$

where all the $\mathbb{h}_i$ are N-dimensional binary vectors. The parity-check matrix $\mathbf{H}$ of an LDPC code is an $(N-K)\times N$ matrix, in which the $i^{\,th}$ row is represented by $\mathbb{h}_i$. Hence, the set of parity-check equations are more compactly written as

$$\mathbf{c}\,\mathbf{H}^T = \mathbf{0}. \tag{2.18}$$

Consequently, a parity-check code is defined by its parity-check matrix $\mathbf{H}$, which performs $M = (N-K)$ parity checks on a received codeword.

An LDPC code is a parity-check code with a parity-check matrix **H** consisting mostly of 0s. For a *regular* LDPC code, **H** contains a fixed number d_c of 1s in each of its columns and $d_r = d_c(N/M)$ of 1s in each of its rows. A regular LDPC code is therefore denoted by a 3-tuple (N, d_c, d_r). The code rate $R = (K/N)$ of an LDPC code is given by $R \leq 1 - (d_c/d_r)$, with the equality holds when **H** is of full rank (Ryan, 2004). If the number of 1s in each column and/or in each row are/is not constant, the code is referred to as an *irregular* LDPC code.

Graph representation Although in Section 2.1, the Tanner graph representation is already applied with the CDO codes, it is elaborated in this section with more details.

Tanner (Tanner, 1981) introduced the concept of constructing long error-correcting codes based on short-length codes and a bipartite graph (known as the Tanner graph). The LDPC codes are regarded as a specific subclass of these codes.

A bipartite graph is a graph in which the nodes can be partitioned into two disjoint classes, with nodes within a same class unconnected to each other. In defining a new long-length code, all the nodes in one class are associated with the symbols in the code, whilst the nodes in the other class correspond to subcodes with lengths exactly the same as their degrees (number of connections). The idea is illustrated in Fig. 2.9, with the circles representing symbol (variable) nodes and the squares representing subcode (constraint) nodes. The labels along with the edges indicate the symbols' positions in the subcodes.

If the subcodes in Fig. 2.9 are single parity-check codes, the graph defines a low-density parity-check code. Consequently, the Tanner graph of an LDPC code consists of N variable nodes and $M = N - K$ constraint nodes, with each variable node representing one column in **H** and each constraint node representing one row. A variable node is connected to a constraint node when the entry at the intersection of their corresponding column and row is 1. Accordingly, d_v and d_c in the parity-check matrix correspond to the degrees of the variable nodes and the constraint nodes, respectively, in the bipartite graph representation. For an irregular code, the degree patterns of a Tanner graph are usually specified by the

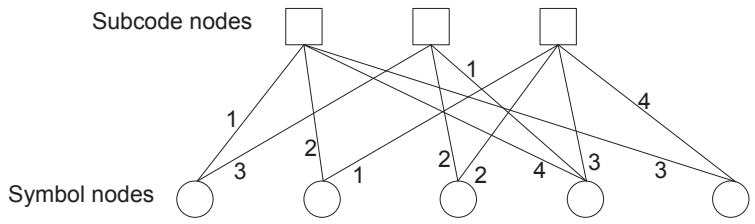


Figure 2.9 Example of the graph representation of a Tanner code.

degree-distribution pairs, which specify the fractions of edges in the Tanner graph connected to nodes of each applicable degree.

2.2.2 Generalized LDPC Codes

The Tanner graph representation (Tanner, 1981) describes the general scheme of constructing long error correcting codes from shorter ones. The choice for the component codes is arbitrary in essence. A series of work (Wang and Fossorier, 2006; Lentmaier and Zigangirov, 1999; Boutros *et al.*, 1999; Miladinovic and Fossorier, 2005) explored this idea and developed a set of LDPC codes with their variable nodes and/or constraint nodes more complex than those stated in the previous section, referred to as the generalized LDPC codes. Fig. 2.10 illustrates the a generalized LDPC code with both its variable nodes and constraint nodes extended, known as the doubly generalized LDPC codes (Wang and Fossorier, 2006). In

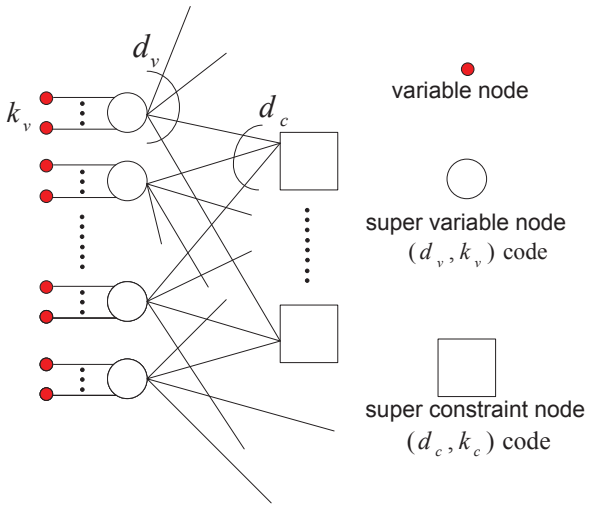


Figure 2.10 A doubly generalized LDPC code.

Fig. 2.10, the solid circles represent the variable nodes according to actual symbols (bits) of the generalized LDPC code ; however, they are not directly connected into the Tanner graph of this code. k_v of these variable nodes are grouped together and connected to a *super variable node* (Wang and Fossorier, 2006). A super variable node is a rate k_v/d_v linear code, which receives k_v input bits from the normal variable nodes (solid circles) and outputs d_v bits, and therefore has a degree of d_v. A *super constraint node*, on the other hand, corresponds to a linear code with code rate k_c/d_c. It has a degree of d_c, i.e., a super constraint node joins d_c connections in the Tanner graph to form a code block. During the decoding process, the messages are transmitted between the super variable nodes and the super constraint nodes,

which resembles the message passing of a normal LDPC code ; however, the actual variable nodes (solid circle) receive messages from the super variable nodes rather than directly from the constraint nodes.

Suppose the number of super variable nodes and super constraint nodes are given by N and M respectively, with $Nd_v = Md_c$; the total number of bits in the doubly generalized LDPC code is given by Nk_v, while the total number of parity bits is given by $M(d_c - k_c)$. Therefore the code rate is lower bounded by

$$1 - \frac{M(d_c - k_c)}{Nk_v} = 1 - \frac{d_v(d_c - k_c)}{k_v d_c}. \tag{2.19}$$

Specifically, for the simple LDPC codes defined in Section 2.2.1, the super constraint nodes are actually repetition codes with $k_v = 1$, and the super variable nodes are single parity-check codes with $d_c - k_c = 1$, (2.19) is thus reduced into $(1 - d_v/d_c)$.

The decoding of generalized LDPC codes may be carried out with maximum *a posteriori* (MAP) decoding at both the super variable node and the super constraint node sides. The generalized LDPC codes lead to sparser Tanner graphs compared with the original LDPC codes and therefore the independence assumption is less compromised as the number of iterations grows large in the decoding process.

2.2.3 The LDPC codes defined over the finite fields

The most well known generalization on the symbol alphabets of the LDPC codes is the employment of the finite fields of order q, i.e., $GF(q)$. The LDPC codes with symbols defined over $GF(q)$ are commonly referred to as the q-ary LDPC codes (Davey and Markay, 1998), which demonstrated superior error performances when compared to the binary LDPC codes of the same bit lengths. This section provides a brief review of the q-ary LDPC codes.

The finite fields A finite field (Galois field) $GF(q)$, is a set of q elements specified by two arithmetic operations : $\oplus$ (addition) and $\otimes$ (multiplication). The specification of the properties of a finite field is given as follows :

- for all $a, b \in GF(q)$, $(a \oplus b), (a \otimes b) \in GF(q)$;
- for all $a, b, c \in GF(q)$, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, $(a \otimes b) \otimes c = a \otimes (b \otimes c)$;
- there exist elements $e_0, e_1 \in GF(q)$ (called identities), such that for all $a \in GF(q)$, $e_0 \oplus a = a \oplus e_0 = a$, $e_1 \otimes a = a \otimes e_1 = a$;
- for each $a \in GF(q)$, there exists $b \in GF(q)$ such that $a \oplus b = b \oplus a = e_0$;
- for each $a \in GF(q)/e_0$, there exists $b \in GF(q)/e_0$ such that $a \otimes b = b \otimes a = e_1$;
- for all $a, b, c \in GF(q)$, $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$.

In the study of the LDPC codes, we are interested in the *characteristic-two finite fields*, i.e., $q = 2^z$ for some integer z. The elements of this set of field may be alternatively represented by their binary images (Lidl and Niederreiter, 1997), which are unique $z \times z$ square matrices.

Consider a degree z polynomial $\mathfrak{R}(D) = a_0 + a_1 D + \cdots + a_{z-1} D^{z-1} + D^z$ with its coefficients $a_i, i = 0, 1, \ldots, z-1 \in GF(q)$, its *order* is the smallest integer e for which $\mathfrak{R}(D)$ divides $D^e + 1$. $\mathfrak{R}(D)$ is *primitive* (Lidl and Niederreiter, 1997) if it has order $e = 2^z - 1$.

Given a primitive polynomial $\mathfrak{R}(D)$, the primitive element in $GF(q)$ is represented by

$$\mathfrak{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & \vdots \\ \vdots & & & \ddots & 1 \\ a_0 & a_1 & a_2 & \cdots & a_{z-1} \end{bmatrix}. \tag{2.20}$$

Furthermore, the elements in the finite field can be written accordingly with $GF(2^z) \leftrightarrow \{\mathbf{0}, \mathbf{I}, \mathfrak{A}, \mathfrak{A}^2, \ldots, \mathfrak{A}^{q-2}\}$, where $\mathbf{0}$ and $\mathbf{I}$ represent the zero matrix and the identity matrix respectively.

Definition of q-ary LDPC codes The q-ary low-density parity-check codes (Davey and Markay, 1998) have their symbols and parity-check matrices defined over $GF(q)$. Notations in binary LDPC codes are reused in our description, where necessary extensions to their definitions are as follows.

Similar to the binary LDPC codes, a q-ary LDPC code is defined by a sparse parity-check matrix $\mathbf{H}$ of size $M \times N$, where the entries are elements in $GF(q)$. A single code block of a q-ary LDPC code contains N symbols in $GF(q)$, K of which are information symbols, i.e., $M = N - K$ parity symbols are included in each block. The column weight of a regular q-ary LDPC code represents the number of non-zero entries in each of the columns in $\mathbf{H}$ and is denoted by d_v, whilst the row weight d_c refers to the number of non-zero entries in each row. The parity-check equations take the same form as in Section 2.2.1, i.e.,

$$\mathbf{c}\,\mathbf{H}^T = \mathbf{0} \tag{2.21}$$

where $\mathbf{c} = (c_1, c_2, \ldots, c_N)$ represents a codeword and c_n $(n = 1, 2, \ldots, N)$ is the nth symbol in $\mathbf{c}$.

Since most of the entries in $\mathbf{H}$ are 0s, we may write the a single parity-check equation as

$$\sum_{i=1}^{d_r} c_i h_i = 0, \tag{2.22}$$

which represents a single constraint node in the Tanner graph representation of a q-ary LDPC code.

Note that the LDPC codes defined over $GF(q)$ with $q = 2^z$ can be viewed as a specific case of the generalized binary LDPC codes. Denote the binary image of h_i by the square matrix $\mathbf{H}_i$ and the symbol c_i by a binary vector $\mathbf{b}_i = (b_{i,1}, b_{i,2}, \cdots, b_{i,z})$, the parity-check equation in (2.22) is rewritten as

$$\sum_{i=1}^{d_r} \mathbf{b}_i \mathbf{H}_i = \mathbf{0}, \tag{2.23}$$

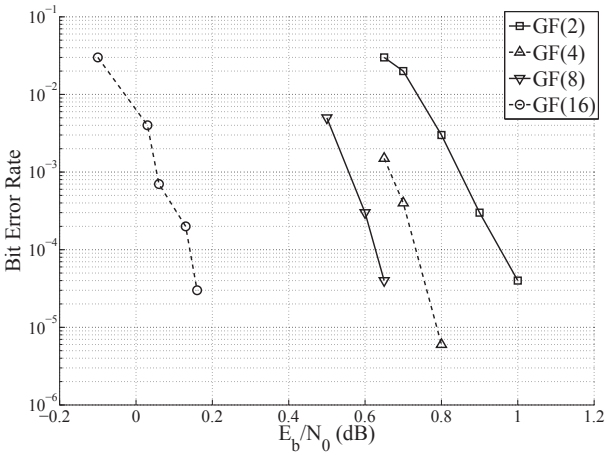
which represents the constraints of a binary, $GF(2)$, parity-check code.

2.2.4 Error performances of q-ary LDPC codes

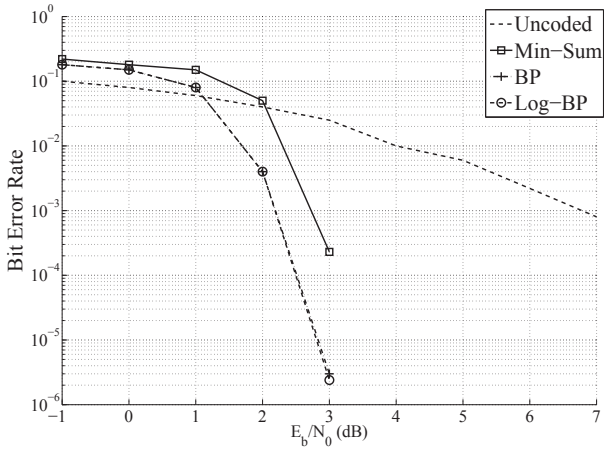
The decoding of the q-ary LDPC codes follows the belief propagation algorithm. Due to the multi-dimensionality of the messages passed in the decoding process, the decoding complexity grows exponentially with the value of q, prohibiting the codes with large values for q from real implementation. Various simplification on the BP algorithm for the q-ary LDPC codes had been proposed in the literature (Wymeersch *et al.*, June 2004; Song and Cruz, 2003; Declercq and Fossorier, 2007; Kschischang *et al.*, 2001).

(Wymeersch *et al.*, June 2004) proposed the Log-domain version of the BP algorithm for the q-ary LDPC codes. Although the proposed algorithm greatly reduced the complexity by eliminating the need for the time-consuming multiplications, the complexity still grows rapidly with q. A further approximated version of the Log-domain decoding was also provided in (Wymeersch *et al.*, June 2004), referred to as the min-sum algorithm. In (Declercq and Fossorier, 2007), the author raised an improved version of the Log-domain decoding; by reducing the number of combinations to be considered in the updating process at the constraint nodes, the author provided a flexible method in the tradeoff between the error performances and the decoding complexity. The fast Fourier transform (FFT)-based decoding algorithm (Kschischang *et al.*, 2001; Song and Cruz, 2003) explores the convolutional property of the calculation at the constraint nodes, and replaces it with the Fourier domain operations whose complexity grows linearly with q, notably accelerating the decoding process without sacrificing the error performances. The Fourier-domain decoding may also find its roots in the hard decoding algorithms (Lin and Costello, 2004) for the BCH codes (Hoc-

quenghem, 1959; Bose and Ray-Chaudhuri, 1960) and the Reed-Solomon codes(Reed and Solomon, 1960). However, as will be explained in Chapter 4, the FFT algorithm requires extremely fine precision, preventing it from real applications.



(a) Error performances of LDPC codes defined over various fields (Davey and Markay, 1998).



(b) Performances of 8-ary LDPC codes under different decoding algorithms (Wymeersch *et al.*, June 2004).

Figure 2.11 Performances of q-ary LDPC codes.

Comparison of the bit error rates under the original BP decoding algorithm for codes defined over different alphabets is given in (Davey and Markay, 1998), which is shown in Fig.2.11(a). It is shown that increasing the value of q helps to improve the error performances of the LDPC codes; i.e., around 0.8 dB improvement could be observed with the code defined over $GF(16)$ as opposed to the binary code at the BER of 3×10^{-5}. Furthermore, the performances of various decoding algorithms are compared in Fig.2.11(b) (Wymeersch *et al.*,

June 2004). It is observed that while the Log-domain algorithm performs approximately the same as the original BP algorithm, the max-Log-SP (min-sum) algorithm compromises error performance for decoding simplicity, i.e., when $E_b/N_0 = 3$ dB, the bit error rate obtained with the min-sum decoder is almost two orders of magnitudes higher than that with the BP decoder.

2.2.5 Construction methods for Tanner graphs of LDPC codes

The principal assumption of the BP decoding algorithm is that the messages passed on the Tanner graphs are independent throughout the entire process. The assumption holds when the additive white Gaussian noise added to different transmitted symbols are independent and the Tanner graph contains no cycles. Therefore, the effort on the construction of LDPC codes had been focused on the optimization of the graph structure.

The parity-check matrices of the original codes of Gallager (Gallager, 1963) are designed by splitting **H** into several sub-matrices ; one of these sub-matrices has specific structures and the rests are its column permutations. These codes enjoy good distance properties, but cycles of length 4 are not excluded from their corresponding Tanner graphs, leading to violation of the independence assumption during the first iteration in the decoding process.

Mackay has developed a series of semi-random algorithms to construct **H** (Mackay, 1999). In general, to achieve Tanner graphs with smaller number of short-length cycles, higher computational complexity is unavoidable.

In (Kou *et al.*, 2001), a construction method using a geometric approach was put forward. Several codes have been constructed based on lines and points of the Euclidean geometry or the projective geometry over the finite fields. The constructed LDPC codes have a girth of at least 6 and can be extended to longer codes using the extension methods presented in (Kou *et al.*, 2001). Several other approaches have been proposed with ideas similar to that in (Kou *et al.*, 2001). For example, constructing **H** based on optical codes and Reed Solomon codes has been proposed in (Djurdjevic *et al.*, 2003).

In (Hu *et al.*, 2001), the author progressively establishes connections between the variable nodes and the constraint nodes in an edge-by-edge manner to avoid short-length cycles, leading to the name progressive edge growth (PEG). Lower bounds on the minimum distance and girth had also been derived in (Hu *et al.*, 2001).

In (Richardson and Urbanke, 2001), Richardson and Luby examined the ensembles of LDPC codes parameterized by the degree-distribution pairs. They evaluated the maximum level of channel distortion, referred to as *threshold*, above which reliable communication cannot be achieved using a typical code in the ensemble. The method, namely the density evolution, is further used to optimize the degree distribution pairs so as to achieve the largest

threshold.

2.3 Summary

In this chapter, we reviewed the definitions of the convolutional doubly orthogonal codes as well as their iterative decoding algorithms. Depending on their encoding structures, the CDO codes are categorized into the non-recursive and recursive subgroups. Generically, the CDO codes may be viewed as another set of graph codes and therefore decoded with the iterative threshold decoding algorithm or the BP decoding algorithm. The doubly orthogonal feature of CDO codes leads to the outstanding error performances of the CDO codes. The recursive codes are more promising when low BERs is required under harsh communication circumstances. Furthermore, the simplified CDO codes serves as attractive candidates when attention is on reducing the decoding latency. Search and construction methods are also briefly covered in this chapter. The search for CDO codes with small memory order has been a concern in the literature, several methods had been proposed regarding the issue. The free distances of the binary convolutional codes constitutes one of the most indicative measure of these codes, which is bounded through various code parameters such as the code rate, constraint length, etc.

The principal concepts of low-density parity-check (LDPC) codes are also covered in this chapter. The LDPC codes were originally defined with parity-check matrices, which had been transformed into the form of bipartite graphs by Tanner. Moreover, we briefly reviewed the construction methods for these codes. Some of these methods, although with different origins, serve the common purpose of reducing the number of short-length cycles in the Tanner graphs. One other method, the density evolution, can be used to find the optimized degree-distribution pairs of a Tanner graph so as to improve the decoding threshold.

The generalization on the LDPC codes in terms of the their alphabets is also discussed in this chapter. Particularly, the characteristic-2 finite fields had been applied as the symbol alphabets. The generalized codes outperform the binary ones in terms of error performances under the belief propagation algorithm ; which is achieved at the cost of increased decoding complexity. In light of the complexity issue, several acceleration algorithms had been proposed in the literature as alternatives to the original BP decoding algorithm for the q-ary LDPC codes (Wymeersch *et al.*, June 2004; Song and Cruz, 2003; Declercq and Fossorier, 2007; Kschischang *et al.*, 2001).

CHAPTER 3

The q-ary Non-recursive CDO Codes with Single-shift-register Structure

This chapter concerns the extension of the binary CDO codes onto the finite fields $GF(q)$. Specifically, we consider the CDO codes with single-shift-register structure adapted to q-ary alphabets. The set of codes are decoded with either the iterative threshold decoding or the BP decoding algorithms which must be adapted to accommodate alphabets over $GF(q)$. In order to make the decoding complexity manageable, as in the BP decoding for q-ary LDPC codes, the Fourier-domain processing (Declercq and Fossorier, 2007) is adopted at each constraint node in the decoding process for the q-ary CDO codes. For this set of codes, the error performances of both decoding algorithms substantially depend on the scaling factor in the decoding process, whose value is selected using simulations.

As shown in the simulation results presented in this chapter, the q-ary CDO codes outperform their binary counterparts in terms of error performances under the iterative threshold decoding algorithm. Specifically, they achieve an additional 0.25 dB coding gain in the waterfall region compared to binary codes with a similar latency. However, the advantage of the q-ary CDO codes is far more pronounced in the error floor region, where the bit error probabilities are about one order of magnitude lower than those of binary codes. Similar to the case of q-ary LDPC codes, these improvements are obtained at the cost of some increased complexity. The decoding complexity for the iterative threshold decoding algorithm is evaluated using the number of different operations in the decoding process which are then compared with those of the binary codes. The q-ary CDO codes achieve still better error performances with the belief propagation algorithm as compared to the iterative threshold decoding ; however, the BP decoding suffers from yet an increased decoding complexity, which is attributed to both the different variable node processing and the increased number of iterations to achieve reasonable error performances. Under both the decoding algorithms, the performances of the simplified q-ary CDO codes are compared with those of the non-simplified q-ary CDO codes. Whilst enjoying a much smaller decoding latency, the error performances of the simplified codes are somehow compromised.

The decoding thresholds for the q-ary CDO codes with single-shift-register structure under the BP decoding algorithm are calculated using the density evolution with the Gaussian approximation (Li *et al.*, 2009) adapted to our codes. Numerical results showed that the decoding thresholds are affected by various code parameters such as the field order q, the number of connections from the shift register J, and the scaling factor a.

3.1 q-ary CDO codes

In this section, we present the definition and encoding structure of the q-ary CDO codes, with $q = 2^z$ for some integer z.

A rate 1/2 q-ary systematic convolutional encoder with a single shift register is shown in Fig. 3.1. $u_t, p_t \in GF(q)$ represent, respectively, the information symbol and the generated parity symbol at time t. Each parity symbol is generated as the linear combination of a finite set of J information symbols specified by their connection positions $\alpha_i, i = 1, 2, \ldots, J$, where $\alpha_1 < \alpha_2 < \ldots < \alpha_J$. Note that each of the memory cells in the shift register stores 1 symbol of z bits of the information symbol sequence. During each time slot t, z bits of source information are input to the shift register, and the symbols stored in the shift register are shifted simultaneously from each memory cell into the next. Just like binary CDO codes, a q-ary convolutional code defined this way is a convolutional doubly orthogonal code if the set $\{\alpha_i\}$ satisfies the conditions as specified in Section 2.1.1.

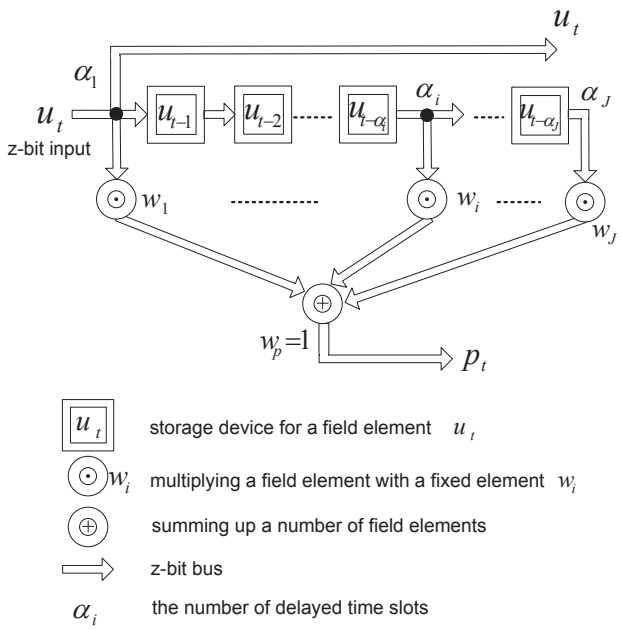


Figure 3.1 Encoder for q-ary CDO codes, $q = 2^z$.

The doubly orthogonal condition assures that the messages in the decoding process are independent for the first two iterations.

Fig. 3.1 depicts the encoder of a q-ary CDO code with a single shift register. As shown in Fig. 3.1, in a q-ary CDO encoder, each connection from its shift register is assigned a specific weight denoted $w_i \in GF(q), i = 1, 2, \ldots, J$; furthermore, the weight assigned to the output

of the adder is denoted $w_p \in GF(q)$. The parity symbol p_t is therefore generated by

$$w_p \cdot p_t = \sum_{i=1}^{J} w_i \cdot u_{t-\alpha_i}, \tag{3.1}$$

where the additions and multiplications are defined over $GF(q)$. Note that the weight vector $(w_1, \ldots, w_J, w_p)$ provides the same generating equation as $(w_1 \cdot w_p^{-1}, w_2 \cdot w_p^{-1}, \ldots, w_J \cdot w_p^{-1}, 1)$, where w_p^{-1} denotes the multiplicative inverse of w_p in $GF(q)$. Without loss of generality, we assume that $w_p = 1$. Hence, a q-ary CDO code is defined by the set of 2-tuples $\{(\alpha_1, w_1), (\alpha_2, w_2), \ldots, (\alpha_J, w_J)\}$, with the encoding equation

$$p_t = \sum_{i=1}^{J} w_i \cdot u_{t-\alpha_i}. \tag{3.2}$$

In Fig. 3.1, the encoding process is performed by first multiplying the symbols stored in the shift register with the weights of their corresponding stages, the parity symbol p_t is then formed by the summation of all the calculated products. Note that here both multiplications and additions are performed in $GF(q)$ arithmetics, which can be implemented either with binary logic circuits or with a simple look-up table.

3.2 Decoding of the q-ary CDO codes

The decoding process for the q-ary CDO codes is conducted with either the iterative threshold algorithm or the belief propagation algorithm. The two are essentially similar except for the following two distinctions :

1. At a variable node, whilst the BP algorithm calculates the exclusive messages for each constraint nodes incident to a variable node, the iterative threshold decoding includes all the incoming messages in the processing, transmitting identical messages to each of the incident constraint nodes ;
2. in each iteration, the BP decoder utilizes the messages calculated from the previous iteration while the iterative threshold decoding employs the messages in their most updated version ; i.e., during the μ th iteration, the BP decoder uses only the messages computed in the $(\mu-1)$ th iteration, however, the iterative threshold decoding algorithm uses the messages calculated in the current iterations if they are available and from the $(\mu - 1)$ th iteration otherwise.

The differences lead to a modest decrease in the computation complexity for the iterative threshold decoding in a single iteration ; however, as will be demonstrated in our simulations,

the iterative threshold decoding requires a much smaller number of iterations to achieve satisfactory error performances than that required by the BP algorithm.

The next section demonstrates the details of both the iterative threshold decoding and the BP decoding algorithms. As indicated in (Kschischang *et al.*, 2001), the decoding is most conveniently carried out with the Fourier-domain messages.

3.2.1 Iterative threshold decoding for the q-ary CDO codes

Regarding (3.2), an information symbol u_t is involved in J distinct parity-check equations :

$$w_j \cdot u_t + p_{t+\alpha_j} + \sum_{i=1, i\neq j}^{J} w_i \cdot u_{t+\alpha_j-\alpha_i} = 0, \tag{3.3}$$

where $j = 1, 2, \ldots, J$. As in the q-ary LDPC codes (Kschischang *et al.*, 2001; Declercq and Fossorier, 2007), in order to avoid the necessity of enumerating all solutions of (3.3), we adopt the fast Fourier transform (FFT)-based approach for our decoding algorithm.The decoding is performed in the probabilistic domain.

First denote the elements of the finite field as

$$GF(q) = \{\gamma_0, \gamma_1, \ldots, \gamma_{q-1}\}. \tag{3.4}$$

Let $\lambda_t(\gamma_k)$ denote the probability that $u_t = \gamma_k$; let $f_t(\gamma_k)$ represent the *a posteriori* probability of the event $u_t = \gamma_k$ given the channel information and similarly, let $g_t(\gamma_k)$ be the *a posteriori* probability of the event $p_t = \gamma_k$. Furthermore, we denote S_j as the event that the jth parity-check equation in (3.3) is satisfied. To simplify the notations, we define the vector $\boldsymbol{\gamma} = (\gamma_{j,p}, \gamma_{j,1}, \gamma_{j,2}, \ldots, \gamma_{j,j-1}, \gamma_{j,j+1}, \ldots, \gamma_{j,J}), \gamma_{j,i} \in GF(q)$, for $u_{t+\alpha_j-\alpha_i} = \gamma_{j,i}$ and $p_{t+\alpha_j} = \gamma_{j,p}$. A set of these vectors $Z_j^{\gamma_k}$ is defined as $Z_j^{\gamma_k} = \{\boldsymbol{\gamma} : w_j \cdot \gamma_k + \gamma_{j,p} + \sum_{i=1, i\neq j}^{J} w_i \cdot \gamma_{j,i} = 0\}$. Therefore, λ_{t,γ_k} is calculated as

$$\begin{aligned}
\lambda_t(\gamma_k) &= f_t(\gamma_k) \cdot \prod_{j=1}^{J} \mathrm{Prob}(S_j | u_t = \gamma_k) \\
&= f_t(\gamma_k) \cdot \prod_{j=1}^{J} \sum_{\boldsymbol{\gamma} \in Z_j^{\gamma_k}} \left(g_{t+\alpha_j}(\gamma_{j,p}) \prod_{i=1}^{j-1} f_{t+\alpha_j-\alpha_i}(\gamma_{j,i}) \prod_{i=j+1}^{J} \lambda_{t+\alpha_j-\alpha_i}(\gamma_{j,i}) \right),
\end{aligned} \tag{3.5}$$

where the operations are conducted in real number arithmetics.

In (3.5), the message $\lambda_t(\gamma_k)$, i.e., the probability that $u_t = \gamma_k$, is the product of the messages from its related parity-check equations concerning γ_k, i.e., the probability that a parity-check equation is satisfied given $u_t = \gamma_k$. It is similar to the BP decoding except that

the calculation in the variable node concerns all the incoming messages, i.e., the processing is not 'exclusive', which reduces the computation complexity. As will be demonstrated in our simulation, the modification also improves the speed of convergence in the decoding compared to that in the BP decoding; therefore, the threshold decoding requires a much smaller number of iterations to successfully decode a symbol. The difficulty in computing (3.5) arises from the fact that the number of vectors in $Z_j^{\gamma_k}$ amounts to q^{J-2}, which rapidly approaches a prohibitively large number as q increases. Therefore, an alternative method is required to carry out the computation in (3.5).

Note that for different γ_k, the constraint in $Z_j^{\gamma_k}$ is a linear equation with fixed coefficients

$$(1, w_1, w_2, \ldots, w_{j-1}, w_{j+1}, \ldots, w_J) \tag{3.6}$$

and variables

$$(p_t, u_{t+\alpha_j-\alpha_1}, u_{t+\alpha_j-\alpha_2}, \ldots, u_{t+\alpha_j-\alpha_{j-1}}, u_{t+\alpha_j-\alpha_{j+1}}, \ldots, u_{t+\alpha_j-\alpha_J}). \tag{3.7}$$

In order to verify whether a vector $\boldsymbol{\gamma}$ is in $Z_j^{\gamma_k}$, a total number of $(J-1)$ multiplications and $(J-1)$ additions over $GF(q)$ is required. To eliminate the need for multiplications when verifying each different $\boldsymbol{\gamma}$, we consider an imaginary shift register output $u'_t = u_t \cdot w_i$, where $i = 1, 2, \ldots, J$. For $u_t = \gamma_l, l = 0, 1, \ldots, q-1$, $u'_t = \gamma_l \cdot w_i$. The corresponding probability for u'_t is therefore $\lambda'_t(\gamma_l \cdot w_i) = \lambda_t(\gamma_l)$. Let $\gamma_l = \gamma_m \cdot w_i^{-1}$ for some $m, m = 0, 1, \ldots, q-1$. When the weight w_i is fixed, all the q γ_m are distinct for different γ_l, and therefore form an enumeration in $GF(q)$. Denote $\boldsymbol{\lambda}_t = (\lambda_t(\gamma_0), \lambda_t(\gamma_1), \ldots, \lambda_t(\gamma_{q-1}))$ as the probability vector for u_t, and denote the probability vector for u'_t by $\boldsymbol{\lambda}'_t = (\lambda'_t(\gamma_0), \lambda'_t(\gamma_1), \ldots, \lambda'_t(\gamma_{q-1}))$. Using γ_l and γ_m as the indices of coordinates in $\boldsymbol{\lambda}_t$ and $\boldsymbol{\lambda}'_t$ respectively, the elements of the two vectors are thus related by $\lambda'_t(\gamma_m) = \lambda_t(\gamma_m \cdot w_i^{-1}), m = 0, 1, \ldots, q-1$. $\boldsymbol{\lambda}'_t$ may therefore be obtained simply by a permutation of coordinates in $\boldsymbol{\lambda}_t$ according to the code's rule of indices $\gamma_l = \gamma_m \cdot w_i^{-1}$. We define a permutation function $\mathbf{P}_{w_i}(\cdot)$ with respect to a specific weight w_i as

$$\mathbf{P}_{w_i}(\boldsymbol{\lambda}_t) = (\lambda_t(\gamma_0 \cdot w_i^{-1}), \lambda_t(\gamma_2 \cdot w_i^{-1}), \ldots, \lambda_t(\gamma_{q-1} \cdot w_i^{-1})), \tag{3.8}$$

where $i = 1, 2, \ldots, J$.

In the iterative decoding process, we denote $\boldsymbol{\lambda}_t^{(\mu)}$ as the probability vector for u_t after the μth iteration. Additionally, $\mathbf{f}_t = (f_t(\gamma_0), f_t(\gamma_1), \ldots, f_t(\gamma_{q-1}))$, $\mathbf{g}_t = (g_t(\gamma_0), g_t(\gamma_1), \ldots, g_t(\gamma_{q-1}))$, and the vector $\boldsymbol{\beta}_{t+\alpha_j-\alpha_i}^{(\mu)}$ is the permutated version of $\boldsymbol{\lambda}_{t+\alpha_j-\alpha_i}^{(\mu)}$ with respect to w_i, i.e., $\boldsymbol{\beta}_{t+\alpha_j-\alpha_i}^{(\mu)} = \mathbf{P}_{w_i}(\boldsymbol{\lambda}_{t+\alpha_j-\alpha_i}^{(\mu)})$. Furthermore, we denote $\otimes$ and $\prod\otimes$ as component-wise multiplications on the real vectors.

Using the above notations in (3.5), the decoding equation becomes

$$\boldsymbol{\lambda}_t^{(\mu)} = \mathbf{f}_t \otimes \prod_{j=1}^{J} \otimes \mathbf{P}_{w_j}^{-1}(\mathbf{M}_j), \tag{3.9}$$

where $\mathbf{M}_j = (m_{j,\gamma_0}, m_{j,\gamma_1}, \ldots, m_{j,\gamma_{q-1}})$, and

$$m_{j,\gamma_k} = \sum_{\boldsymbol{\gamma} \in X_j^{\gamma_k}} \left(g_{t+\alpha_j}(\gamma_{j,p}) \prod_{i=1}^{j-1} \beta_{t+\alpha_j-\alpha_i}^{(\mu-1)}(\gamma_{j,i}) \prod_{i=j+1}^{J} \beta_{t+\alpha_j-\alpha_i}^{(\mu)}(\gamma_{j,i}) \right). \tag{3.10}$$

Note that all operations are conducted in real number arithmetics.

The set $X_j^{\gamma_k} = \{\boldsymbol{\gamma} : \gamma_k + \gamma_{j,p} + \sum_{i=1, i\neq j}^{J} \gamma_{j,i} = 0\}$ is applied in replacement of $Z_j^{\gamma_k}$ due to the permutation operation performed. With the help of the permutation function, the operations in (3.10) constitute a $(J-1)$-fold convolution of the vectors $\mathbf{g}_{t+\alpha_j}$, $\boldsymbol{\beta}_{t+\alpha_j-\alpha_i}^{(\mu-1)}$ for $i = 1, 2, \ldots, j-1$ and $\boldsymbol{\beta}_{t+\alpha_j-\alpha_i}^{(\mu)}$ for $i = j+1, j+2, \ldots, J$. Therefore, as in q-ary LDPC codes, it is convenient to apply the FFT over finite field (Declercq and Fossorier, 2007) in our algorithm to eliminate the enumeration in $X_j^{\gamma_k}$. The operations of the decoding algorithm may be summarized as

$$\boldsymbol{\lambda}_t^{(\mu)} = \mathbf{f}_t \otimes \prod_{j=1}^{J} \otimes \mathbf{P}_{w_j}^{-1} \left\{ \text{iFFT} \left[\text{FFT}(\mathbf{g}_{t+\alpha_j}) \otimes \prod_{i=1}^{j-1} \otimes \text{FFT}(\boldsymbol{\beta}_{t+\alpha_j-\alpha_i}^{(\mu-1)}) \prod_{i=j+1}^{J} \otimes \text{FFT}(\boldsymbol{\beta}_{t+\alpha_j-\alpha_i}^{(\mu)}) \right] \right\}, \tag{3.11}$$

where

$$\boldsymbol{\beta}_{t+\alpha_j-\alpha_i}^{(\mu)} = \mathbf{P}_{w_i}(\boldsymbol{\lambda}_{t+\alpha_j-\alpha_i}^{(\mu)}). \tag{3.12}$$

Here FFT$(\cdot)$ and iFFT$(\cdot)$ denote the fast Fourier transform (FFT) and inverse FFT of a vector with finite field indices. The frequency-domain decoding also has its roots in the BCH codes and the Reed-Solomon (RS) codes over finite fields (Lin and Costello, 2004).

In our simulation, we adopted the decoding equation as stated in (3.13),

$$\begin{aligned} \boldsymbol{\lambda}_t^{(\mu)} \;=\; & \mathscr{T}_t^{(\mu)} \mathbf{f}_t \otimes T_a \Bigg(\prod_{j=1}^{J} \otimes \mathbf{P}_{w_j}^{-1} \Bigg\{ \mathscr{S}_{t,j}^{(\mu)} \text{iFFT} \Bigg[\text{FFT}(\mathbf{g}_{t+\alpha_j}) \otimes \\ & \prod_{i=1}^{j-1} \otimes \text{FFT}(\boldsymbol{\beta}_{t+\alpha_j-\alpha_i}^{(\mu-1)}) \prod_{i=j+1}^{J} \otimes \text{FFT}(\boldsymbol{\beta}_{t+\alpha_j-\alpha_i}^{(\mu)}) \Bigg] \Bigg\} \Bigg), \end{aligned} \tag{3.13}$$

where $\mathscr{S}_{t,j}^{(\mu)}$ is selected such that the sum of all the coordinates of the vector in the braces is 1; similarly, $\mathscr{T}_t^{(\mu)}$ ensures that $\sum_{k=0}^{q-1} \lambda_t^{(\mu)}(\gamma_k) = 1$. The function $T_a(\cdot)$ is introduced in order to reduce the impact of messages from later iterations in order to improve the decoding

performance, which is discussed in our simulations. For a vector $\mathbf{v} = (v_1, v_2, \ldots)$, $T_a(\mathbf{v}) = (v_1^a, v_2^a, \ldots)$, for $a \in (0, 1)$. Finally, after a total number of N iterations, the information symbol u_t is decoded using

$$\hat{u}_t = \underset{\gamma_k}{\operatorname{argmax}} \lambda_t^{(N)}(\gamma_k). \tag{3.14}$$

The decoding process in a single round of iteration is summarized in Fig. 3.2. The message vectors are transformed between the real domain and the Fourier domain in each iteration, reducing the sum-operations into component-wise multiplications.

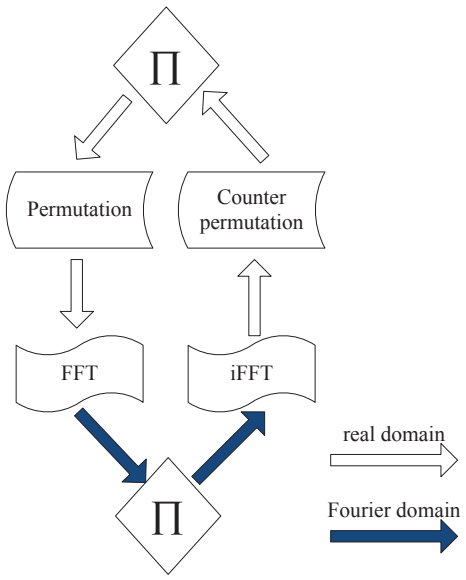


Figure 3.2 Decoding process.

3.2.2 BP decoding for the q-ary CDO codes

The belief propagation decoding algorithm for the q-ary CDO codes with single-shift-register structure is similar to the threshold decoding process. However, in each iteration, the decoder only utilizes the messages at the previous iteration; furthermore, the updating rule at the variable node excludes the message from the edge through which the updated message is to be returned, and hence the decoding process cannot be summarized in a single equation as in (3.13). The details of the BP decoding is provided as follows.

The message on a variable node representing a parity symbol p_t is never updated in the BP algorithm, therefore we only need to define the outgoing message from the information symbol node u_t. The message from a variable node u_t to the j^{th} constraint node is defined as

$$\boldsymbol{\lambda}_{t,j} = (\lambda_{t,j}(\gamma_0), \lambda_{t,j}(\gamma_1), \ldots, \lambda_{t,j}(\gamma_{q-1})) \tag{3.15}$$

for $j = 1, 2, \ldots, J$; and

$$\boldsymbol{\nu}_{t,j} = (\nu_{t,j}(\gamma_0), \nu_{t,j}(\gamma_1), \ldots, \nu_{t,j}(\gamma_{q-1})) \tag{3.16}$$

represents the message coming from the $j^{\,th}$ constraint node incident to this variable node.

We now describe the principal operations :

(a) Initialization

At the inception of the decoding process, the message from a variable node at the $0^{\,th}$ iteration is assigned the value calculated with the channel information, i.e.,

$$\boldsymbol{\lambda}_{t,j}^{(0)} = \mathbf{f}_t, \tag{3.17}$$

for $j = 1, 2, \ldots, J$.

(b) Constraint node updating

During the $\mu^{\,th}$ iteration, the $j^{\,th}$ constraint node calculates the message to be transmitted to the variable node corresponding to an information symbol u_t using

$$\begin{aligned} \boldsymbol{\beta}_{t+\alpha_j-\alpha_i,j}^{(\mu-1)} &= \mathbf{P}_{w_i}(\boldsymbol{\lambda}_{t+\alpha_j-\alpha_i,j}^{(\mu-1)}) \\ \boldsymbol{\nu}_{t,j}^{(\mu)} &= \mathbf{P}_{w_j}^{-1}\left\{ \mathscr{S}_{t,j}^{(\mu)} \text{iFFT}\left[\text{FFT}(\mathbf{g}_{t+\alpha_j}) \otimes \prod_{i=1, i\neq j}^{J} \otimes \text{FFT}(\boldsymbol{\beta}_{t+\alpha_j-\alpha_i,j}^{(\mu-1)}) \right] \right\}, \end{aligned} \tag{3.18}$$

where the variable $\mathscr{S}_{t,j}^{(\mu)}$ is again selected such that the sum of all the coordinates of the vector in the braces is 1.

(c) Variable node updating

The message calculated by a variable node u_t to its $j^{\,th}$ constraint node in the $\mu^{\,th}$ iteration is

$$\boldsymbol{\lambda}_{t,j}^{(\mu)} = \mathscr{T}_t^{(\mu)} \mathbf{f}_t \otimes T_a \left(\prod_{i=1, i\neq j}^{J} \otimes \boldsymbol{\nu}_{t,i}^{(\mu)} \right), \tag{3.19}$$

where the coefficient $\mathscr{T}_t^{(\mu)}$ ensures that $\sum_{k=0}^{q-1} \lambda_{t,j}^{(\mu)}(\gamma_k) = 1$. Note that no updating is needed for a variable node representing a parity symbol for it has only one connected constraint node whose message is excluded from the calculation.

(d) Decision

After a total number of N iterations, the message vector for an information symbol u_t is calculated using

$$\boldsymbol{\lambda}_t = \mathscr{T}_t^{(N)} \mathbf{f}_t \otimes T_a \left(\prod_{i=1}^{J} \otimes \boldsymbol{\nu}_{t,i}^{(N)} \right), \tag{3.20}$$

and the information symbol u_t is decoded into

$$\hat{u}_t = \underset{\gamma_k}{\operatorname{argmax}} \lambda_t(\gamma_k). \tag{3.21}$$

The appropriate number of iterations (N) needed to achieve reasonable error performances is obtained through computer simulations. In Section 3.3, it is shown that the iterative threshold decoding requires much smaller number of iterations than the BP decoding to achieve satisfactory error performances for our codes.

3.2.3 Decoding complexity

In this section, the decoding complexity of the q-ary CDO codes are compared to that of the binary CDO codes in terms of the number of various operations applied in the decoder.

Table 3.1 Number of operations needed to decode 1 bit in 1 iteration under the iterative threshold decoding algorithm.

	Additions	Multiplications	Table look-ups	Comparisons
q-ary CDO codes	J^2q	$\frac{J^2q}{z}$	$\frac{J^2q}{z}$	-
Binary CDO codes	1	-	-	$2J^2$

The decoding of the q-ary CDO codes concerns z-bit symbols rather than single bits as in binary CDO codes. For fair comparisons among codes with different values of q, we consider the decoding complexity per bit rather than per symbol in a single iteration. Table 3.1 shows the numbers of different operations in the iterative threshold decoding algorithm to calculate the message on a single bit in 1 iteration for q-ary CDO codes with $q > 2$ as compared to the Log-domain threshold decoding for the binary CDO codes (Cardinal *et al.*, 2003). Note that the additions, multiplications, and comparisons are performed on real values, whilst the table look-ups are conducted to perform $GF(q)$ arithmetics. The decoding algorithm for q-ary CDO codes are somewhat more complex than that of binary codes; the numbers of real-valued operations are of order $O(J^2q)$. However, in actual applications, the decoder is usually implemented with received quantized messages (discrete message alphabets). Under this scenario, the additions and multiplications of the messages may be implemented easily

with look-up tables (LUTs), consuming substantially less time than the arithmetic operations with real values. Although the time consumed by different operations varies and depends on the digital processing system used ; in this thesis, we assume for simplicity that various operations consume the same amount of time. Detailed comparisons over the complexity issue is presented in our simulations.

The decoding complexity for the BP decoding algorithm is comparable with the iterative threshold decoding. The operations required for the processing of the message over a single edge is the same for both the variable node and constraint node updating process, although the iterative decoding process concerns the most updated incoming messages while the BP decoder uses messages only from the previous round. However, the BP decoding process is still somehow more complex than the iterative threshold decoding owing to the variable node updating process : the BP decoder calculates for each edge incident to a variable node a distinct message while the iterative threshold decoder transmits identical messages over all the edges incident to the same variable node, moderately affecting the complexity.

3.3 Simulation results

In this section, we present the simulation results on the q-ary CDO codes under both the iterative threshold decoding and the BP decoding algorithms. Information symbols are generated randomly from the elements in $GF(q)$ and fed into the shift register to produce the code sequence, which is transmitted through the AWGN channel before being captured by the receiver. The model of the transmission system is depicted in Fig. 3.3. The simulations have been carried out assuming binary phase shift keying (BPSK) with coherent decoding. The encoded sequence over $GF(q)$ are mapped into bits before they are sent to the BPSK modulator. Let $(b_k^1, b_k^2, \ldots, b_k^z)$ denote the binary representation of $\gamma_k \in GF(q = 2^z)$. Define $\mathfrak{A}_1(y) = 1/[1+\exp(2y/\sigma^2)]$, and $\mathfrak{A}_0(y) = 1/[1+\exp(-2y/\sigma^2)]$, where σ denotes the standard deviation of the noise. f_{t,γ_k} and g_{t,γ_k} are hence calculated as

$$f_{t,\gamma_k} = \prod_{i=1}^{z} \mathfrak{A}_{b_k^i}(y_{t,i}^u) \quad \text{and} \quad g_{t,\gamma_k} = \prod_{i=1}^{z} \mathfrak{A}_{b_k^i}(y_{t,i}^p), \tag{3.22}$$

for $k = 0, 1, \ldots, q-1$. Here $y_{t,i}^u$ and $y_{t,i}^p$ denote the ith received signal for u_t and p_t respectively.

The receiver applies the iterative decoding algorithms as described in Section 3.2. The scaling parameter a is selected through simulations in such a way that it provides the best error performances for our codes in the E_b/N_0 region under concern. Note that we applied the pipelined decoding architecture as in (Jimenez Felstrom and Zigangirov, 1999), i.e., each symbol is decoded with a fixed number of iterations before its value is determined by hard

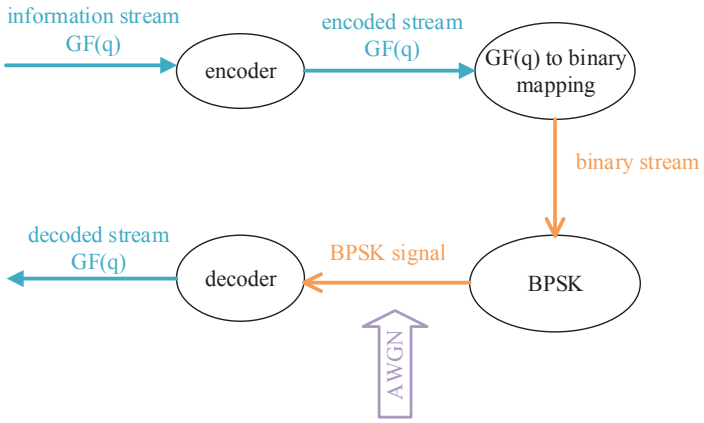


Figure 3.3 Transmission system model.

decision.

The simulations using the q-ary CDO codes have been conducted for different q, J and a values. The connection positions $\{\alpha_i\}$ in our codes are the same as in (Cardinal *et al.*, 2003; Haccoun *et al.*, 2005).

A note on the weights

The selection of the weights $\{w_i\}$ exerts phenomenal impacts on the error performances of the q-ary CDO codes. For each code, we select one set of weights uniformly randomly from the non-zero elements of $GF(q)$ in constructing the code. The random selection proved to be advantageous over the identical weight selection in terms of BERs. The difference in the error performances of the two weight selection scheme has its origin in the iterative decoding processes of the q-ary codes. In Chapter 5, we discuss in detail the mechanism through which the weight selection affects the decoding process; meanwhile, we briefly present the conclusion of the analysis in Chapter 5 as follows :

- The random selection has the effect of diluting the erroneous messages when they are propagated in short-length cycles in the Tanner graph.
- The weight on an edge determines the impact of its connected variable node on other variable nodes involved in the same parity-check equation; in the BP decoding, the random selection scatters the erroneous information onto different coordinates of those messages involved in the same constraint node.
- a q-ary code with identical weights is equivalent to the binary codes under the BP decoding algorithm, eliminating the benefits of increasing the size of the message alphabets.

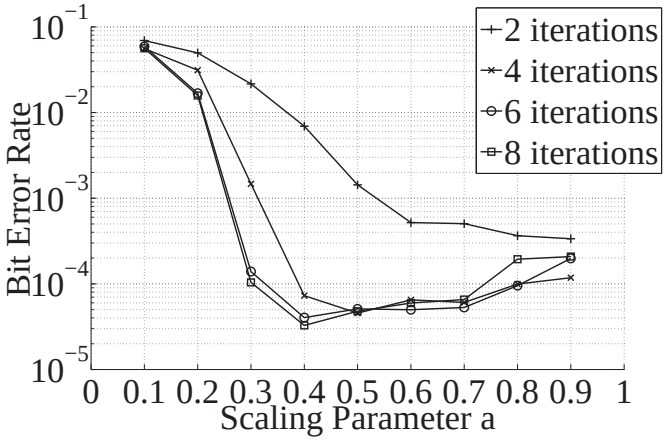
3.3.1 Error performances under the iterative threshold decoding

Due to the doubly orthogonal properties of the codes, the independence among the messages is guaranteed for the first two iterations in the decoding of CDO codes. Starting from the third iteration, the messages become dependent. The parameter a in the function $T_a(\cdot)$ ameliorates this effect by weighing down the impact of the messages from the constraint nodes. In our first set of simulations, we aim to find the optimal values of a for various q values under typical values of E_b/N_0. Specifically, for $J = 7$ and $q = 8$ and 16, the optimal values of a at $E_\mathrm{b}/N_0 = 3$ dB are determined using computer simulations and chosen as those minimizing the error probabilities, for different iteration numbers. From Fig. 3.4, it clearly appears that the optimal values of a are equal to 0.4 for both $q = 8$ and $q = 16$. The curves in Fig. 3.4 are obtained at $E_\mathrm{b}/N_0 = 3$ dB, but other simulation results verified that $a = 0.4$ is a reasonable choice for a wide range of E_b/N_0 values. For a comparison, in (Cardinal *et al.*, 2003), the value of the scaling factor for the binary CDO codes was set to 0.2.

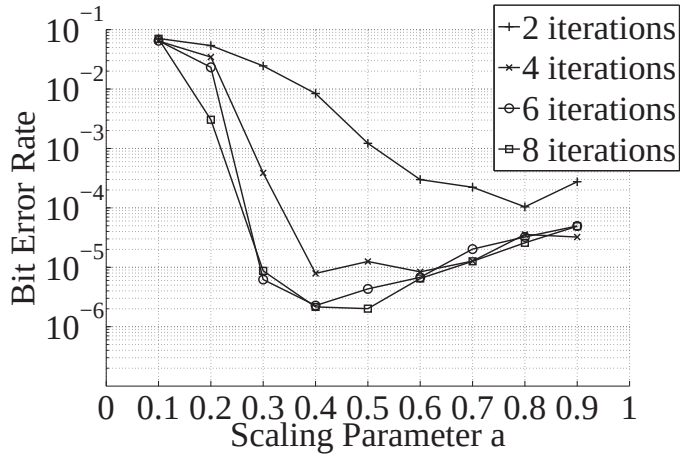
It is also worth mentioning that applying the same scaling factor a for all the iterations is not the optimal scaling scheme for the iterative threshold decoding. As mentioned in (Cardinal *et al.*, 2003) for the binary codes, it is beneficial to apply different scaling factor for each iteration rounds; however, the process of determining the best set of scaling factors is itself a topic requiring exploration. In our simulation, we apply simplest scheme; i.e., the scaling factor a remains the same for all the iterations.

Following the determination of the value of a, we further compare the bit error rates of q-ary CDO codes with scaled ($a = 0.4$) and the unscaled ($a = 1$) schemes in Fig. 3.5. It is observed that the scaled version ($a = 0.4$) of the decoder substantially outperforms non-scaled decoder, except for the 2-iteration case. This exception is attributed to the fact that the independence among messages in the first 2 iterations is guaranteed; and therefore with a larger a, the algorithm tends to converge faster. However, for subsequent iterations the messages become dependent. For a large value of a (e.g., $a = 0.7, 0.8$), the inaccurate messages from the parity checks may force the decoder to converge to a locally preferred symbol value, leading to unsatisfactory performance. Therefore, it is preferable to apply the scaled version of the iterative threshold decoding in actual applications.

We further compare the error performances of the q-ary CDO codes with different q and J values in Fig. 3.6. It is observed that for the same J, a larger value of q leads to a lower error floor, e.g., when $J = 10$, 16-ary codes has an error floor almost 1-order of magnitude lower than that of 8-ary codes after the 6th or 8th iteration. Similar to the q-ary LDPC codes, the differences in the error floor region may be attributed to the locally dense structure of q-ary CDO codes in the iterative decoding process. However, we may also observe that with large values for q, the error performance at the waterfall region is compromised.

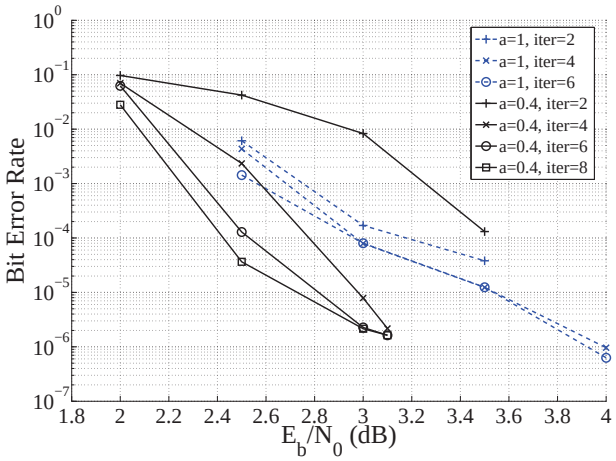


(a) $J = 7$,$q = 8$,and $E_{\mathrm{b}}/N_0 = 3$dB.

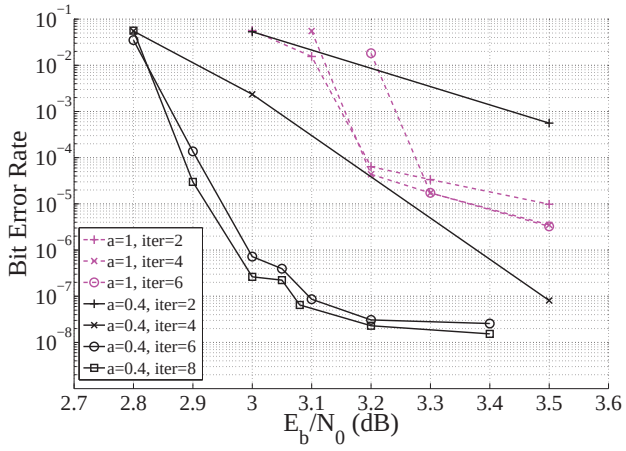


(b) $J = 7$,$q = 16$, and $E_{\mathrm{b}}/N_0 = 3$dB.

Figure 3.4 BER as a function of a for $J = 7$, $q = 8, 16$, and different number of iterations in the threshold decoding.



(a) $J = 7$,$q = 16$.



(b) $J = 10$,$q = 16$.

Figure 3.5 BER as a function of E_b/N_0 for $J = 7, 10$, $q = 16$, $a = 0.4, 1$, and different number of iterations.

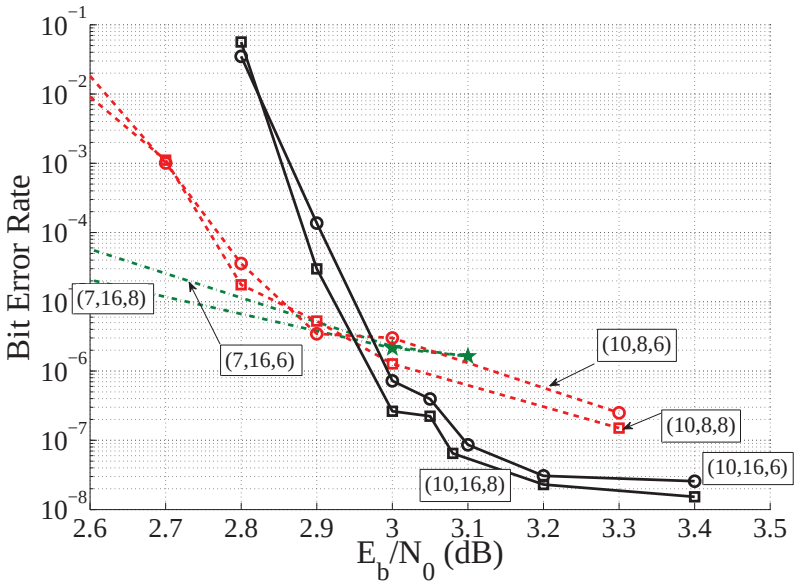


Figure 3.6 BER as a function of E_b/N_0 for different (J,q, iteration no.) with $a = 0.4$ in the threshold decoding.

The error performances of the q-ary CDO codes are closely related to the number of iterations performed. As shown in Fig. 3.6, applying 8 iterations in the decoding has no dramatic improvement over the case of 6 iterations, which can be owing to the fact that the messages in the threshold decoder are not 'exclusive' as in the BP decoder; i.e., in (3.13), the calculation of $\boldsymbol{\lambda}_t^{(\mu)}$ involves all its related J parity-check equations. We may conclude that it is quite sufficient to conduct 8 iterations in the decoding process for these codes, although fewer than 8 iterations may also be quite sufficient in practice.

Table 3.2 List of best known α_J of CDO and SCDO codes for different J.

J	CDO	SCDO
7	222	82
10	1698	340
15	23193	-

The error performances of our codes are compared with those of some known binary CDO codes with a single shift register (Haccoun *et al.*, 2005; Roy, 2006) in Fig. 3.7. Note that the decoding latency, which is proportional to $(\alpha_J \times z \times N)$ for a q-ary CDO code, is of primary concern in our study. The connection positions on the shift registers for our codes are the same as those of the binary codes in (Haccoun *et al.*, 2005; Cardinal *et al.*, 2009), whilst the weights along the connections are selected uniformly randomly from the non-zero elements of

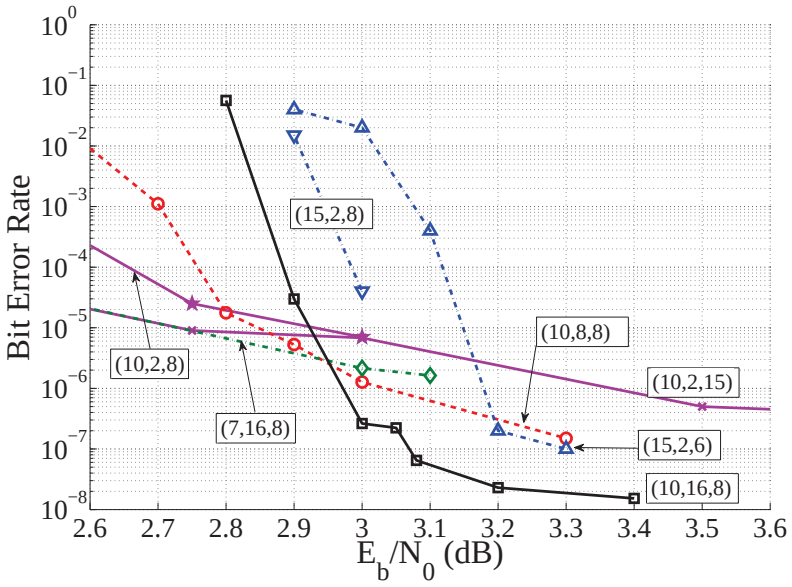


Figure 3.7 BER as a function of E_b/N_0 for different (J,q, iteration no.) compared to binary codes with $J = 10$ (Roy, 2006) and $J = 15$ (Haccoun *et al.*, 2005) codes under the threshold decoding.

$GF(q)$. The best known values of α_J for different values of J are provided in (Haccoun *et al.*, 2005) and (Cardinal *et al.*, 2009), from which we list those under our concern in Table 3.2. The listed values for α_J are the smallest obtained in (Haccoun *et al.*, 2005) for fixed values of J. It is desirable to apply these values since the decoding latency of the CDO codes is proportional to α_J. In practice, if a CDO code with a large span is required, we may exercise one of the two options : (a) select a large J so as to obtain a large α_J; (b) for a small J, search for connection positions with a large α_J, leading to a CDO code with a sparser Tanner graph as compared to the code obtained in (Haccoun *et al.*, 2005) with the same J (and potentially performs better in terms of BERs).

The $(J = 10, q = 8)$ $(\alpha_J = 1698)$ code performs comparably with the $(J = 15, q = 2)$ $(\alpha_J = 23193)$ binary code (Haccoun *et al.*, 2005) in the error floor region, and outperforms the binary code in the waterfall region after 8 iterations (0.25 dB gain). We may note that the $(J = 10, q = 8)$ code is not as competitive as the $(J = 10, q = 2)$ binary code when E_b/N_0 is relatively low. However, if we compare the $(J = 10, q = 2)$ binary code with the $(J = 7, q = 16)$ $(\alpha_J = 222)$ code, we observe that with the same number of iterations, i.e., 8 iterations, the 16-ary code outperforms the binary code by approximately 0.2 dB in all of the E_b/N_0 range under concern. The $(J = 7, q = 16)$ code with 8 iterations even achieves a lower error floor than the $(J = 10, q = 2)$ binary code with 15 iterations. On the aspect of latency, the latency of the $(J = 10, q = 8)$ code is proportional to $1698 \times 3 \times 8 = 40752$

bits as compared to $23193 \times 8 = 191352$ bits for the $(J = 15, q = 2)$ binary code, and the latency of the $(J = 7, q = 16)$ code is proportional to $222 \times 4 \times 8 = 7104$ bits as compared to $1698 \times 15 = 25470$ bits for the $(J = 10, q = 2)$ binary code. In summary, a q-ary CDO code with a relatively small J may achieve comparable error performance as a binary code with a larger J, and therefore enjoy a smaller latency due to the rapid increase of α_J with J.

We may note that the improvements in error performance for q-ary CDO codes always has the by-product of increased complexity due to higher dimension of the messages vector $\boldsymbol{\lambda}_t^{\mu}$. As an example, according to Table 3.1, the number of operations needed to decode 1 bit in 1 iteration for the $(J = 10, q = 8)$ code is approximately 5 times that for the $(J = 15, q = 2)$ code. However, with the $(J = 10, q = 8)$ code, we achieve a coding gain of 0.25 dB in the waterfall region after 8 iterations and a smaller latency (around 1/5) as compared to the $(J = 15, q = 2)$ code. Similarly, we may observe that the $(J = 7, q = 16)$ code has 5 times the complexity for each bit and 1/4 the latency compared with those of the $(J = 10, q = 2)$ code while it achieves 0.2 dB gain over the E_b/N_0 range under concern after the 8th iteration.

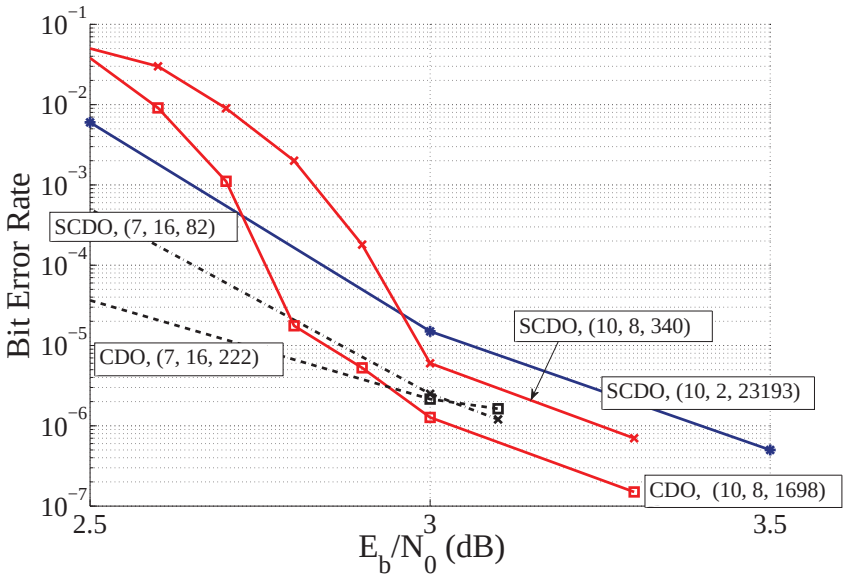


Figure 3.8 BER as a function of E_b/N_0 for CDO and SCDO codes with different (J, q, α_J), with 8 iterations in the threshold decoding.

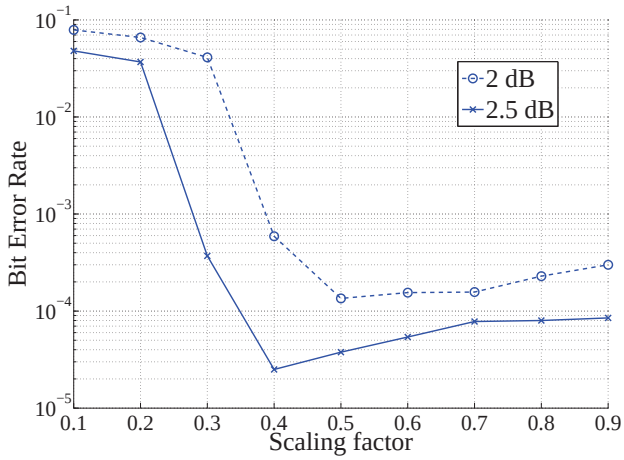
Additionally, a set of simulations are performed to examine the influence of the extension of SCDO codes to finite fields in Fig. 3.8. It is observed that similar to binary codes, using SCDO codes sacrifices slightly in error performance for large improvement in the latency. When $J = 10$, the 8-ary SCDO code suffers from around 0.2 dB loss in performance at high E_b/N_0 region with approximately 1/5 of the latency of the 8-ary CDO codes (340 to

1680). When $J = 7$, the 16-ary SCDO codes suffers from 0.2 dB loss in error performance when compared to the 16-ary CDO codes when E_b/N_0 is low, but the two codes performs comparably at high E_b/N_0 region. However, the $(J = 7, q = 16)$ SCDO codes enjoys a latency of around 1/3 as that of the $(J = 7, q = 16)$ CDO code (82 to 222). Furthermore, the error performances of the q-ary SCDO codes are compared to the $(J = 10)$ binary SCDO code in (Cardinal *et al.*, 2009). By comparing the $(J = 10, q = 8)$ and $(J = 10, q = 2)$ SCDO codes, we may observe that just as with CDO codes, the extension to q-ary alphabets has the effect of lowering the error floor. It may also be observed that the $(J = 7, q = 16)$ SCDO code enjoys a 0.25 dB gain over the $(J = 10, q = 2)$ SCDO code in all of the E_b/N_0 range under concern with a similar latency, i.e., $82 \times 4 \times 8 = 2688$ bits to $340 \times 8 = 2720$ bits. Therefore, we may conclude that the extension of binary codes to finite fields is also applicable for simplified CDO codes, with similar benefits as the extension for CDO codes. In actual applications, the SCDO codes are preferred over the CDO codes when decoding latency is of concern, for they compromise little in the error performances whilst greatly reducing the decoding latency; however, the CDO codes are still advantageous when smaller BERs are required, especially when E_b/N_0 is relatively low.

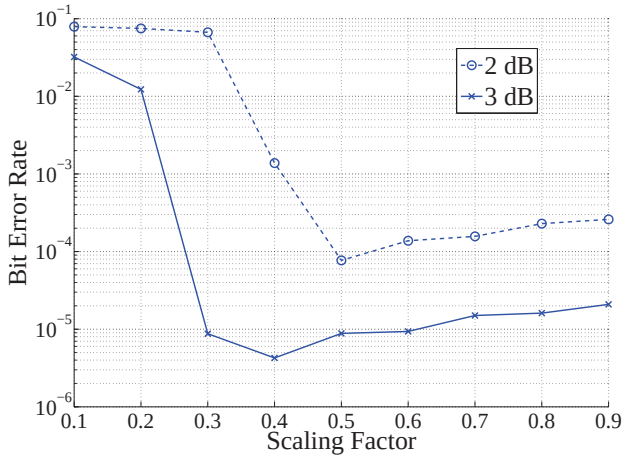
3.3.2 Error performances under the BP decoding

Similar to the cases with the iterative threshold decoding (Shen *et al.*, 2013), firstly we optimize the scaling factor a in the sense of minimizing the BERs at the last iteration in the decoding process for various CDO codes specified by (J, q) values. For each of the codes under consideration, we select 2 typical values of E_b/N_0 and examine the bit error rates under different scaling factor a. Fig. 3.9 demonstrates an example of the selection of a for the codes with $J = 7$ after 50 iterations.

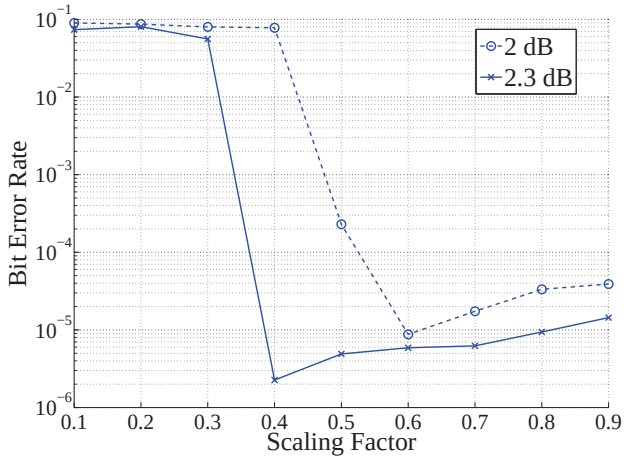
Unlike in the case with iterative threshold decoding (Shen *et al.*, 2013), for most of the codes under concern, the choice of a is E_b/N_0 dependent, i.e., different working E_b/N_0 requires different values of a to achieve satisfactory error performances. For instance, in Fig. 3.9(c), the optimal choice of a for the (7, 16) CDO codes when $E_b/N_0 = 2.3$ dB is 0.4; whilst applying $a = 0.4$ for $E_b/N_0 = 2$ dB results in a BER around 10^{-1} at the last iteration. The same phenomenon has been observed for most of the codes in our simulation. When E_b/N_0 is relatively small, a larger value for a is desirable to raise the impact of the constraint-node messages in order to overwhelm the unsatisfactory initial message of a variable node calculated from the channel information; on the other hand, when E_b/N_0 increases, channel message becomes more reliable and a smaller a serves to reduce the impact of sub-optimal messages from the constraint nodes resulted from the short-length cycles in the graphs. Although it is more desirable to select different values for a under different E_b/N_0, we select for each



(a) $J = 7$,$q = 4$;



(b) $J = 7$,$q = 8$;



(c) $J = 7$,$q = 16$;

Figure 3.9 BER as a function of a for CDO codes with $J = 7$ different and q value.

code in our simulation an appropriate value for the most typical E_b/N_0 values, as listed in Table 3.4.

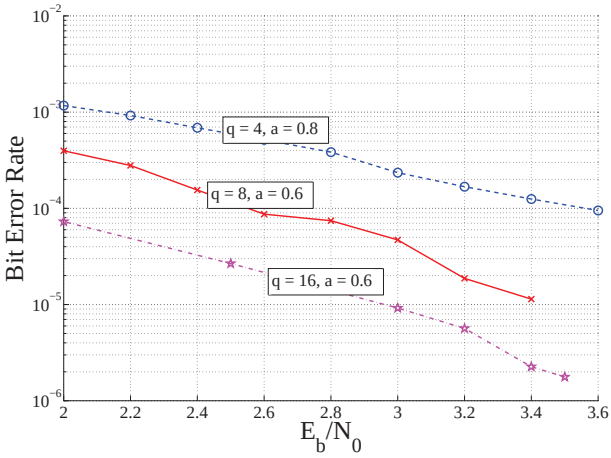
Table 3.3 Selected scaling parameter a for codes with different (J, q) values

	$J = 5$	$J = 7$	$J = 10$
$q = 4$	0.8	0.5	0.5
$q = 8$	0.6	0.5	0.5
$q = 16$	0.6	0.5	1

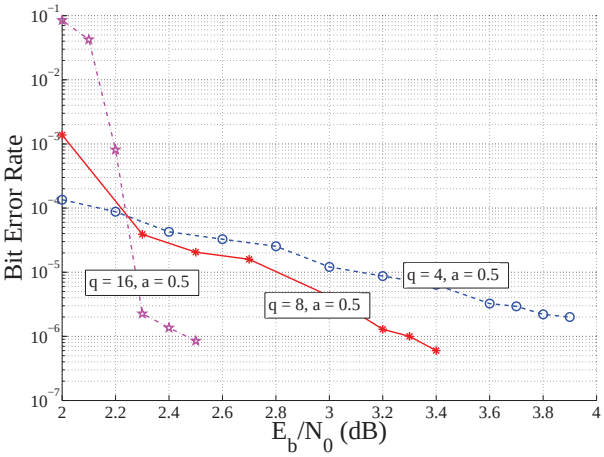
In Table 3.4, it is observed that for codes with identical q value, the optimal value of the scaling parameter a decreases with J, with an exception for the $(10, 16)$ code. This is because with larger number of connections, a variable node receives messages from an abundance of sources (a large number of constraint nodes) which may help its processing, making it unnecessary to apply a large a to emphasize these messages; on the other hand, a smaller value of a helps to ameliorate the negative impact from the violation of the independence assumption caused by short-length cycles in their Tanner graphs. However, when both J and q are large, e.g., the $(10, 16)$ code, it is hard for the decoder to converge with a small value of a due to that with a large number of connections and coordinates along each connection, it becomes unlikely for the 'correct' message to dominate the decoding process if the message is diluted by a small a; furthermore, when the 'correct' messages are relatively insignificant in the decoding, the message corrupted by cycles in the decoding process has more deleterious effect since it is propagated through multiple edges.

The error performances of CDO codes with $J = 5, 7$ and 10 with different q values with the scaling factor a selected as in Table 3.4 are presented in Fig. 3.10. It is observed that for CDO codes with the same J, larger q values generally leads to lower error rates when E_b/N_0 is relatively higher. As demonstrated in Fig. 3.10(b), as $E_b/N_0 =$ gets large, the $(7, 16)$ code demonstrate significant advantage over the other two codes, with approximately 0.8 dB gain over the $(7, 8)$ code at the BER of 10^{-6}. An exception is with the bit error performance of the $(10, 16)$ code in Fig. 3.10(c), which enjoys an approximately identical waterfall as the $(10, 8)$ code but compromised performance in the error floor region. It is mainly attributed to the large value of a applied, i.e. $a = 1$. A large a serves to accelerate the convergence in lower E_b/N_0 region by emphasizing the message from the constraint nodes, while exaggerating the negative impact of the short-length cycles when E_b/N_0 is relatively large.

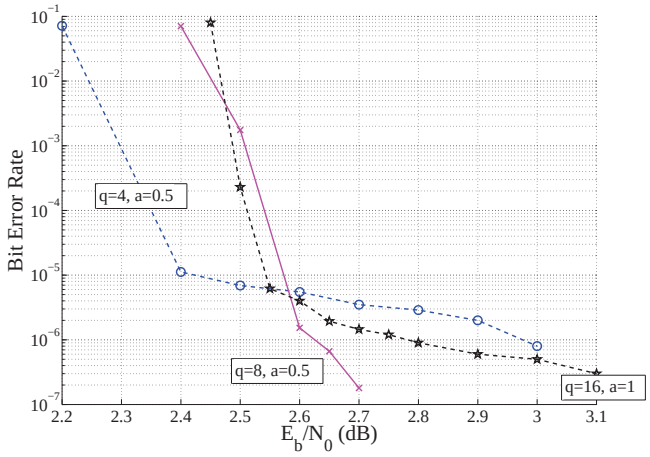
The BER curves of some of the CDO codes under both the BP algorithm and the iterative threshold algorithm are plotted in Fig. 3.11, where the curves for the iterative threshold decoding are taken from (Shen *et al.*, 2013). It is shown that under most circumstances, the BP decoding outperforms the threshold decoding for the E_b/N_0 under concern. For instance,



(a) $J = 5$;



(b) $J = 7$;



(c) $J = 10$;

Figure 3.10 BER as a function of E_b/N_0 for CDO codes with $J = 5, 7$, and 10 with different q values.

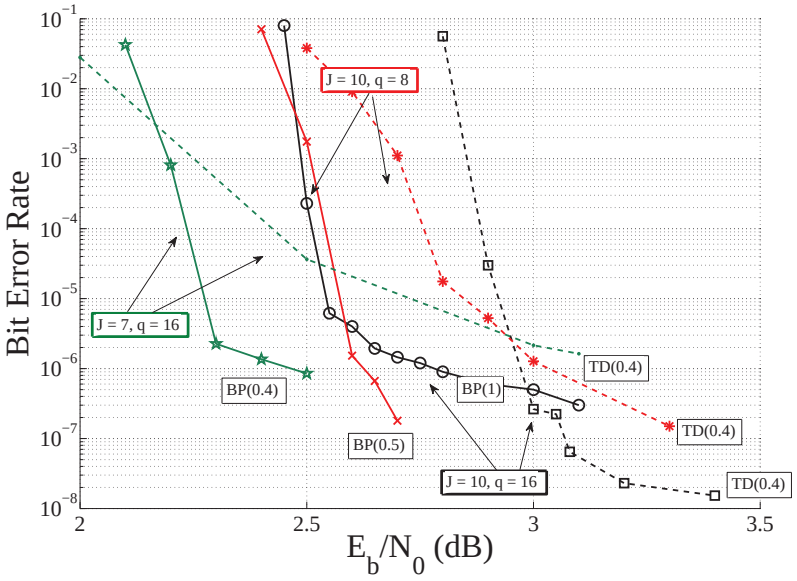
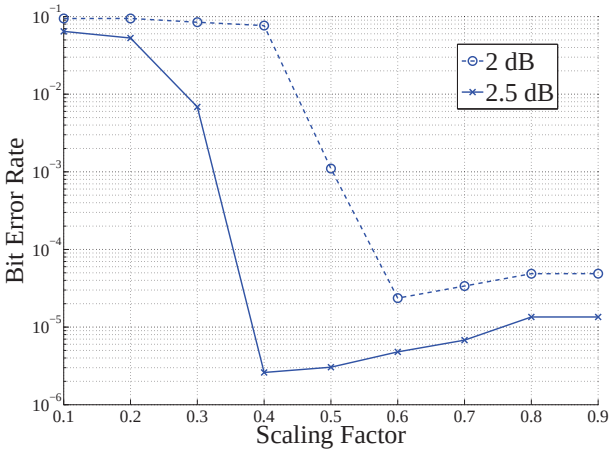


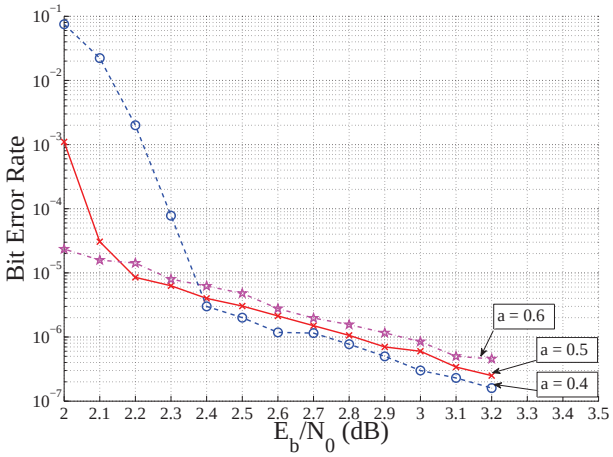
Figure 3.11 BER as a function of E_b/N_0 for CDO codes with belief propagation decoding and threshold decoding.

for the $(10, 8)$ code, the BP decoder enjoys a gain of more than 0.5 dB at the bit error rate of 10^{-7} as compared to the iterative threshold decoding. An exception, however, is again with the $(10, 16)$ code; while the code demonstrates an earlier waterfall performance with the BP decoding, the performance in the error floor region is compromised as opposed to threshold decoding, i.e., the BER with the BP decoder is about 1 order of magnitude higher that with the threshold decoder at the error floor. This exception, could also be attributed to the large value of a applied with the BP decoder for the $(10, 16)$ code; a large value of a accelerates the convergence of the decoder in moderate BER region whilst exaggerating the influence of cycles in the Tanner graph when the BER is lower. Therefore, we suppose that if the $(10, 16)$ code is to be applied with E_b/N_0 greater than 2.5 dB, a smaller value of a is required. It is also worth noting that although the BP decoding is advantageous to the threshold decoding in most cases, it entails much larger number of iterations as compared to the latter, i.e., while the BP algorithm requires 50 iterations to achieve reasonable performances in our simulation, iterative threshold decoding applies only 8, owing to its faster convergence resulted from its unique message updating process.

The error performances of some simplified CDO (SCDO) codes are also examined with the belief propagation decoding. Fig. 3.12 to 3.14 illustrate the effects of the scaling parameter a upon the BERs for typical E_b/N_0 values and the BER curves for the best few choices of a for the $(7, 16)$, $(10, 8)$, and $(10, 16)$ SCDO codes.

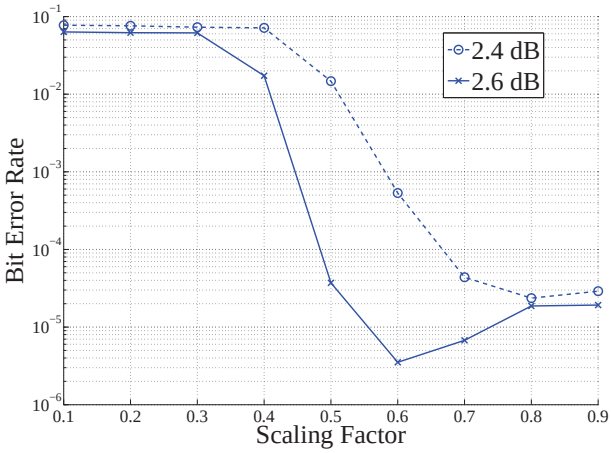


(a) Scaling;

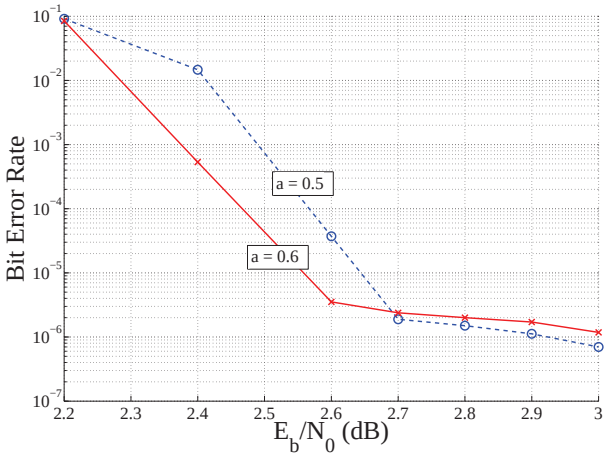


(b) Bit error rate;

Figure 3.12 BER as a function of E_b/N_0 for SCDO codes with $J = 7$ and $q = 16$.

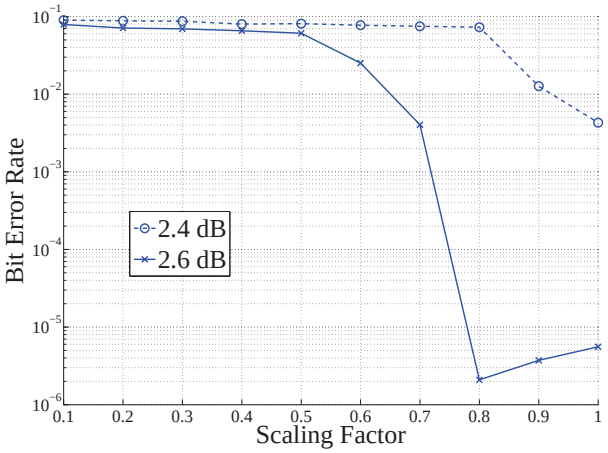


(a) Scaling;

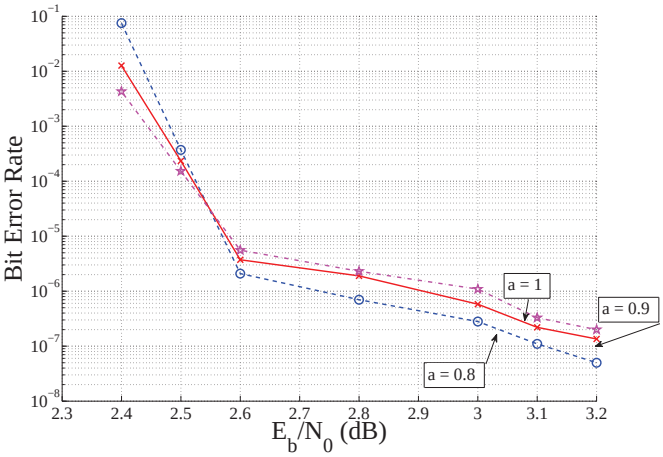


(b) Bit error rate;

Figure 3.13 BER as a function of E_b/N_0 for SCDO codes with $J = 10$ and $q = 8$.



(a) Scaling;



(b) Bit error rate;

Figure 3.14 BER as a function of E_b/N_0 for SCDO codes with $J = 10$ and $q = 16$.

For all the SCDO codes under concern, whilst small variations of a lead to phenomenally different BER performances when E_b/N_0 is relatively low, no obvious difference has been observed in higher E_b/N_0 region; for instance, for the $(7, 16)$ SCDO code, when $E_\mathrm{b}/N_0 >$ 2.4 dB, decoders with $a = 0.4, 0.5, 0.6$ perform almost identically. It is also worth noting that for the $(10, 16)$ code, the BP decoder performs approximately equally with $a = 0.8, 0.9$ and 1. In general, we may conclude that the BP decoder for SCDO codes are less sensitive to the variation in a as opposed to that for the CDO codes, especially when E_b/N_0 is large. Finally, the error performances of the SCDO codes are compared to those of the CDO codes in Fig. 3.15 under the BP decoding. As illustrated in Fig. 3.15, SCDO codes are competitive when E_b/N_0 is relatively low, i.e., they provide almost the same error performances as the CDO codes with the same (J, q) value; however, their performances decay swiftly as compared to the latter in large E_b/N_0 region.

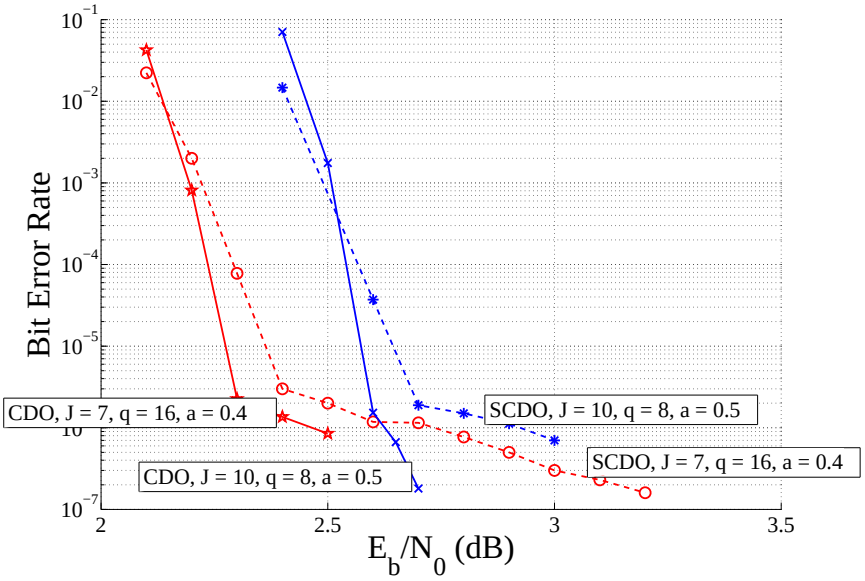


Figure 3.15 BER as a function of E_b/N_0 for SCDO codes and CDO codes with the same (J, q) value under BP decoding.

3.4 Density evolution with Gaussian approximation for non-recursive CDO codes

In this section, we evaluate the decoding thresholds of the q-ary CDO codes using the density evolution method (Richardson and Urbanke, 2001). In the process of the threshold decoding algorithm, the messages from a variable node are dependent, i.e., a variable node transmits identical messages for all its incident constraint node. Hence, it is inappropriate to apply the density evolution for the iterative threshold decoding. The belief propagation decoding algorithm is considered instead in our computation for the decoding thresholds. Note that the non-recursive CDO codes discussed in this chapter is featured by variable nodes with degree of either J or 1. The messages for the degree-1 variable nodes are never updated in the BP algorithm, and hence have no influence on the error probabilities of these codes.

3.4.1 The density evolution

The main difficulty in the density evolution of the q-ary codes lies in the complexity of calculating the probability density functions (p.d.f.s) of the q-dimensional messages. In our calculation, we follow the algorithm in (Li *et al.*, 2009) which applies the Gaussian approximation over the messages.

In the following, we briefly describe the density evolution algorithm and adapt it to our q-ary CDO with the BP algorithm.

It is most convenient to focus on the Log-domain of the messages in the discussion. The density evolution algorithm examines the mean of the Log likelihood ratios (LLRs) for the messages calculated at the variable nodes. The LLR of a message in the decoding of q-ary codes is a $(q-1)$-dimensional vector defined as $\mathbf{s} = (s_1, s_2, \ldots, s_{q-1})$, where

$$s_i = \ln \frac{\lambda(\gamma_0)}{\lambda(\gamma_i)}, \tag{3.23}$$

for $i = 1, 2, \ldots, q-1$. Denote the mean of the message from a variable node after the $\mu^{\,th}$ iteration as $\mathbf{m}^{(\mu)}$, and that from a constraint node as $\mathbf{l}^{(\mu)}$. The updating process for a constraint node of degree d_c is defined as (Li *et al.*, 2009)

$$\mathbf{l}^{(\mu-1)} = \mathcal{G}^{-1}\left(\left[\mathcal{C}_{q-1}(\mathbf{m}^{(\mu-1)})\right]^{d_c-1}\right) \cdot \tilde{\mathbf{1}}_{q-1}, \tag{3.24}$$

where the function $\mathcal{C}_{q-1} : \mathcal{R}^{q-1} \mapsto \mathcal{R}$ is defined as

$$\mathcal{C}_{q-1}(\mathbf{m}) = \frac{1}{q-1}\tilde{\mathbf{1}}_{q-1}^{T} E[\mathcal{F}(\mathbf{s})]; \tag{3.25}$$

and $\mathcal{G}(\cdot) : \mathcal{R} \mapsto \mathcal{R}$ takes the form :

$$\mathcal{G}(x) = \mathcal{C}_{q-1}(x \cdot \tilde{\mathbf{1}}_{q-1}). \tag{3.26}$$

Here $E(\cdot)$ represents the expectation, $\mathcal{F}(\cdot)$ is the mapping from LLR messages to the Fourier domain messages, and $\tilde{\mathbf{1}}_{q-1}$ denotes a $(q-1)$-dimensional all-1 vector. Note that under the assumption that the weights along the edges are selected uniformly randomly from the non-zero elements in $GF(q)$, the mean of the output message from a constraint node is a $(q-1)$-dimensional vector with all-identical coordinates. Only under this assumption, there exists $\mathcal{G}^{-1}(\cdot)$.

Note that for the q-ary CDO codes based on single-shift-register structure, each constraint node is connected to exactly $(J+1)$ variable nodes; J of which are variable nodes corresponding to information symbols, and the remaining one corresponds to a parity symbol. As mentioned earlier, due to that parity variable nodes have degree 1, their messages are not updated throughout the decoding process. Therefore, in our calculation, (3.24) is modified into

$$\mathbf{l}^{(\mu-1)} = \mathcal{G}^{-1}\left(\left[\mathcal{C}_{q-1}(\mathbf{m}^{(\mu-1)})\right]^{J-1} \cdot \mathcal{C}_{q-1}(\mathbf{m}^{(0)})\right) \cdot \tilde{\mathbf{1}}_{q-1}, \tag{3.27}$$

where $\mathbf{m}^{(0)}$ is the mean of the message calculated from the channel information.

The mean of the messages of the variable nodes corresponding to information symbols is updated using

$$\mathbf{m}^{(\mu)} = a(J-1)\mathbf{l}^{(\mu-1)} + \mathbf{m}^{(0)}, \tag{3.28}$$

where a is the scaling factor in our decoding algorithm. Although it is expected that applying the scaling factor should only have negative effects on the decoding thresholds for the messages are weakened as compared to those in a non-scaled decoder, in real applications, the scaling factor is still needed to the combat the dependence among the messages. It is therefore of interest to examine the decoder with $a < 1$ in order to examine the implication of the decoding thresholds on our simulated BER curves for the q-ary non-recursive CDO codes under BP decoding algorithm.

Under the assumption that all-zero codewords are transmitted, the error probability of the CDO codes approaches 0 if $\mathbf{m}^{(\mu)}$ approaches infinity with μ.

In (Li *et al.*, 2009), the authors developed an approximation that calculates $E[\mathcal{F}(\mathbf{s})]$ recursively using the $\phi(\cdot)$ function defined in (Chung *et al.*, 2001b), which approximates $E[\mathcal{F}(\mathbf{s})]$ of a single dimension using exponential functions. However, the approximation becomes less accurate with increasing q.

3.4.2 Numerical results on decoding thresholds

In this section, we present the thresholds calculated using the above algorithm for various J and q values. For the various codes considered in our simulation, their decoding thresholds are determined to be the smallest E_b/N_0 value such that the resulting $\mathbf{m}^{(\mu)}$ approaches infinity with μ.

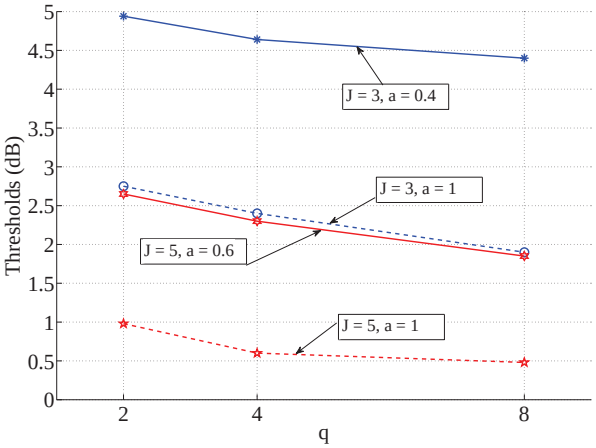
For each of the codes under concern, two decoding thresholds are calculated corresponding to the scaling factor a optimized in Section 3.3 and $a = 1$ (no scaling), as listed in Table 3.4. Note that no valid results was obtained for $q > 8$, due to the inaccuracy in calculating $\mathcal{G}^{-1}(\cdot)$ with the recursive approach in (Li *et al.*, 2009). It is observed that the parameters J, q, and a exert mixed influences on the decoding thresholds of the CDO codes. When no scaling is applied ($a = 1$), for $J = 5$, the threshold decreases from 0.62 dB to 0.48 dB when q increases from 4 to 8, which is attributed to the locally dense structure of codes with a larger value for q; however, the relation is quite the reverse as we increase J, owing to that a large number of connections $(J + 1)$ to the a constraint node together with more options (q) on each connection prevent the decoder from converging to a single decision. When scaling is applied, i.e., when $a < 1$, the thresholds for each of the codes increases for that the messages from the constraint nodes are diluted in the variable node updating process, leading to a slow convergence. However, scaling is still needed in real application to provide a lower error floor, which is not accounted for in the process of density evolution.

Table 3.4 Thresholds (dB) obtained with Gaussian approximation for different J, q, and a values.

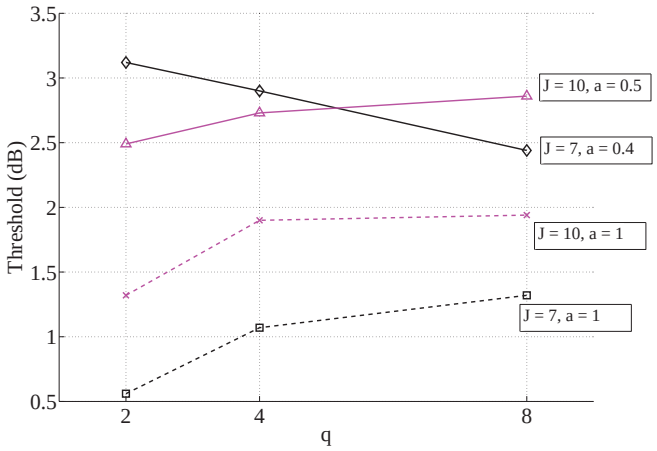
J	$q = 4$	$q = 8$
5	1.40(0.8)	1.91(0.6)
	0.62(1)	0.48(1)
7	2.93(0.4)	2.44(0.4)
	1.07(1)	1.32(1)
10	2.73(0.5)	2.86(0.5)
	1.90(1)	1.94(1)

In order to examine various factors affecting the error performances of the CDO codes, the decoding thresholds of various codes are plotted in Fig. 3.16 as a function of q for different J and a values. When J is relatively small, i.e., $J = 3, 5$, the decoding threshold decreases with q regardless of the value for a; whilst for large J value, e.g., $J = 10$, it increases with q. The dependence of threshold over q for medium J, i.e., $J = 7$, is determined by the value of a; a smaller value for a renders the threshold to be an increasing function of q whilst a large a leads to a decreasing function. It may also be noted from Fig. 3.16 that the threshold

is not a monotonic function of J with other parameters fixed. For instance, for $q = 8$ and $a = 1$, the minimum threshold is observed with $J = 5$. In summary, the decoding thresholds of the q-ary CDO codes based on single-shift-register structure is affected by several factors mixed together : J, q, and a ; change in any single factor alone does not provide sufficient information to determine its effect on the decoding thresholds of these codes under the BP decoding algorithm.



(a) $J = 3, 5$;



(b) $J = 7, 10$;

Figure 3.16 Threshold as a function of q for various codes and a values.

The impacts of these factors could be described as follows :

– J determines the number of connections to the nodes in a Tanner graph. When J is small, each node is getting less help from the neighboring nodes resulting in an unsatisfactory threshold ; when J increases, we observe improved thresholds for the decoder is providing more useful information to a single node ; however, when J further

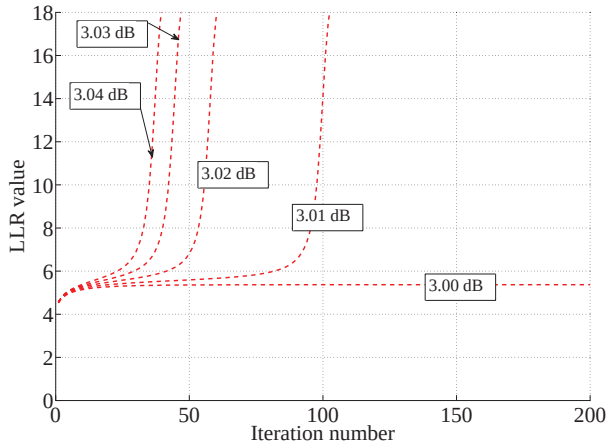
increases, the constraint nodes have relatively large amount of information to consider, making it hard for the 'correct' message to dominate the decoding process, leading to slower convergence and hence increased threshold values.

- q characterizes the way a symbol is checked in a parity-check equation. Since a parity-check equation over $GF(q = 2^z)$ corresponds z parity-check equations in $GF(2)$, each bit in a symbol is checked by more constraints when q is large. Therefore, large q value helps the decoding when J is small by increasing the information along the edges in the Tanner graph without affecting the graph structure; on the other hand, as J gets large, large values of q exaggerates the problem of slow convergence as mentioned above, resulting in increased thresholds.
- a determines the extent that the decoder emphasizes the messages from the constraint nodes, and therefore affects the impact that q and J exert on the thresholds. For instance, when $J = 7$, the messages from the constraint nodes converges slowly due to relatively large J value, therefore the threshold is an increasing function over q when $a = 1$, i.e., when these messages are emphasized; however, when $a = 0.4$, i.e., when the constraint node messages are weakened relative to the channel message, the variable nodes are not getting enough help from the inflow information and therefore large q value decreases the thresholds by providing more information on each single edge.

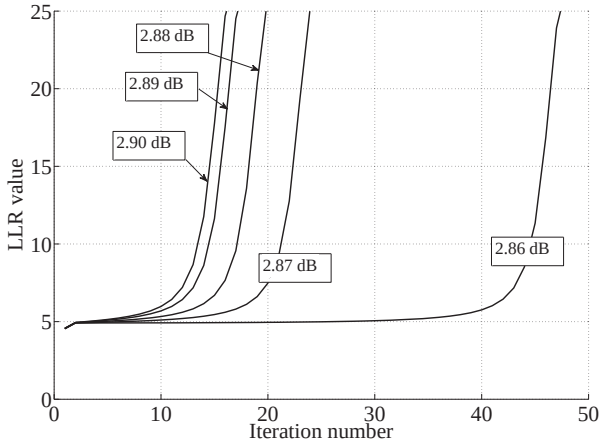
The evaluated decoding thresholds shed lights on the error performances of the q-ary non-recursive CDO codes. For instance, in Fig. 3.10, for the $J = 5$ codes, increasing q improves the error performances in the entire E_b/N_0 region under concern for their decoding thresholds decrease as q gets large; however, for codes with large values of J, e.g., when $J = 7, 10$, the decoding thresholds increases as q increases, i.e., the codes with smaller values of q enjoys better error performances at the waterfall region, leading to the 'crossover' of the BER curves for the $J = 7, 10$ codes.

Finally, the evolution of messages from information variable nodes as a function of the number of iterations are examined in Fig. 3.17. The mean of the message from a variable node is a $(q-1)$-dimensional vector, we only need to examine the first coordinate $m^{(\mu)}(\gamma_1)$ against the iteration number μ. This is because in (3.28), $\mathbf{l}^{(\mu-1)}$ is a vector with all equal coordinates, and $\mathbf{m}^{(0)}$ is the mean of the message calculated from the channel information; furthermore, the binary representation of γ_1 in $GF(q)$ has weight 1 since γ_1 is represented by the binary vector $(1, 0, 0, \ldots)$, which means $m^{(0)}(\gamma_1)$ is the smallest coordinate in $\mathbf{m}^{(0)}$. Hence, $m^{(\mu)}(\gamma_1)$ approaching infinity guarantees the other coordinates to approach infinity, i.e., the error probability approaches 0. In Fig. 3.17(a), it is observed that for those E_b/N_0 values below the threshold, the message from a variable node converges to a finite value as the number of iterations gets large, and therefore the error probability is bounded away from

0; whilst for E_b/N_0 greater than or equal to the threshold, the messages approaches infinity after a 'flat' region. The length of the flat region decreases with increasing E_b/N_0 value, e.g., under 3.01 dB, the decoder requires about 80 iterations to climb out of the flat region while under 3.04 dB, it only takes around 20 iterations. It is also worth noting from Fig. 3.17(b) that under all E_b/N_0 above the threshold, the message requires less than 50 iterations to approach infinity, and therefore 50 iterations are quite sufficient in our decoding process.



(a) $(10, 4)$ CDO code with $a = 0.4$;



(b) $(10, 8)$ CDO code with $a = 0.5$;

Figure 3.17 $m^{(\mu)}(\gamma_1)$ as a function of μ for different codes.

3.5 Summary

In this chapter, we have extended the binary CDO codes with single-shift-register structure onto the finite fields $GF(q)$. Under the iterative threshold decoding algorithm adapted

to the new alphabet, the q-ary CDO codes achieve superior error performances as opposed to the binary ones, especially in the error floor region. The belief propagation algorithm is also adapted to embrace the q-ary CDO codes; while enjoying potentially better error performances than the iterative threshold decoding algorithm, the BP algorithm suffers from slower convergence, requiring a larger number of iterations before the decoder is capable of performing reasonably. Moreover, both the decoding algorithms entails increased complexity as compared to the decoding process of the binary CDO codes, which is the cost of the improved performance. However, the q-ary CDO codes manage to achieve comparable error performances with binary codes with larger number of connections from the shift registers; serving to reduce the decoding latency, which is proportional to the memory order of the shift register. The error performances of the simplified q-ary CDO codes are also investigated in comparison with the CDO codes; as in the cases with the binary codes, the simplified CDO codes slightly compromise the error performances for much smaller memory order, offering an attractive alternative when decoding latency is of primary concern. The density evolution with the Gaussian approximation is adapted for the q-ary CDO codes in the calculation of their decoding thresholds under the BP decoding algorithm. Rather than monotonic over the value of q, the decoding thresholds of the q-ary CDO codes exhibit mixed dependence on various code parameters, such as the number of connections, the field order, and the scaling factor applied in the decoder, etc.

In our discussion on the q-ary non-recursive CDO codes so far, we have applied weights uniformly randomly selected from the non-zero elements of $GF(q)$. In the computer simulations presented in Chapter 5, the random selection prove to be advantageous over selecting identical weights for all the connections. This phenomenon has its roots in the iterative decoding processes, which would be investigated in Chapter 5.

CHAPTER 4

The q-ary Recursive CDO Codes Based on Protographs

In Chapter 3, we extended the non-recursive CDO codes with single-shift-register structure onto the finite fields. On the other hand, the recursive CDO (RCDO) codes (Cardinal *et al.*, 2008) had been shown to provide substantially better error performances than non-recursive ones when decoded with the belief propagation (BP) algorithm. In (Roy *et al.*, 2010), it is also demonstrated that RCDO codes may be regarded as a set of time-invariant low-density parity-check (LDPC) convolutional codes.

In this chapter, we extend the binary recursive convolutional doubly orthogonal (RCDO) codes (Cardinal *et al.*, 2008; Roy *et al.*, 2010) onto the finite fields $GF(q)$. Specifically, we consider RCDO codes defined as time-invariant LDPC convolutional codes adapted to the q-ary alphabets, which have the same constraint lengths (in terms of the number of finite field symbols) as the binary codes in (Roy *et al.*, 2010). The simulation results of the codes of rate $R = b/c = 1/2$ reported in this chapter demonstrate that the extension onto q-ary alphabet contributes to improve the error performances of the RCDO codes as it did for the non-recursive CDO codes. The most prominent effect is that lower bit error rates at high E_b/N_0 region are observed with higher-order fields. Specifically, doubling the alphabet size has the effect of lowering the bit error rate (BER) by approximately one order of magnitude. Down to a BER of 10^{-7}, we have not witnessed error floors for the RCDO codes in our simulations; however, codes with large q values tend to have steeper waterfall performance, indicating better error performances at higher E_b/N_0 than those with smaller q values. Relatively short-memory-length q-ary RCDO codes may thus be used to achieve comparable error performance of longer binary codes, leading to decreased decoding latency. In addition, substantial improvements in error performances are observed when comparing our codes to a typical q-ary LDPC block code with code length comparable to the constraint lengths of the RCDO codes (C.Davey, 1999). Just as in the q-ary LDPC codes, the improvements from using larger q values are obtained at the cost of increased complexity. The new set of codes is therefore advantageous in applications with lower restrictions on the decoding complexity. The error performances of this set of codes under the quantized BP decoding algorithm are also examined through computer simulations in order to consider their practical applications.

4.1 The q-ary recursive CDO codes

In this section, we provide the definition and protograph representation of the q-ary recursive convolutional doubly orthogonal codes.

At time t, a rate b/c q-ary RCDO code receives b information symbols as a vector $\mathbf{u}_t = \{u_t^0, u_t^1, \ldots, u_t^{b-1}\}$ as its input and outputs c code symbols, represented by the vector $\mathbf{v}_t = \{v_t^0, v_t^1, \ldots, v_t^{c-1}\}$, where $u_t^i, v_t^i \in GF(q)$. We only consider systematic encoders in our study, i.e., $v_t^i = u_t^i, i = 0, 1, \ldots, (b-1)$. In the output vector, the $(c-b)$ parity symbols are calculated with the parity-check equations :

$$h_{i,i-b} \times v_t^i = \sum_{j=0}^{b-1} h_{j,i} \times v_{t-\alpha_{j,i}}^j + \sum_{\substack{j=b \\ j \neq i-b}}^{c-1} h_{j,i} \times v_{t-\alpha_{j,i}}^j, \tag{4.1}$$

for $i = b, b+1, \ldots, c-1$. Note that all the arithmetic operations in (4.1) are performed in $GF(q)$. The ith symbol is connected to the $\alpha_{i,j}^{th}$ memory cell of the j^{th} shift register with connection weight $h_{i,j} \in GF(q)$. A general encoder of the q-ary RCDO codes is depicted in Fig. 4.1, which is realized with the observer canonical form (Lin and Costello, 2004). Each of the parity symbol is assigned a shift register, storing appropriately delayed versions of the input information symbols and the feedback parity symbols.

The memory order m of the encoder is defined as

$$m = \max_{\substack{i=0,1,\ldots,c-1 \\ j=0,1,\ldots,c-b-1}} \alpha_{i,j}. \tag{4.2}$$

Similar to the case in Chapter 2, it is convenient to apply the D-transform representation on the parity-check matrix of the RCDO codes :

$$\mathbf{H}^T(D) = \begin{bmatrix} h_{0,0}D^{\alpha_{0,0}} & \ldots & h_{0,c-b-1}D^{\alpha_{0,c-b-1}} \\ h_{1,0}D^{\alpha_{1,0}} & \ldots & h_{1,c-b-1}D^{\alpha_{1,c-b-1}} \\ \vdots & & \vdots \\ h_{c-1,0}D^{\alpha_{c-1,0}} & \ldots & h_{c-1,c-b-1}D^{\alpha_{c-1,c-b-1}} \end{bmatrix}, \tag{4.3}$$

where D represents the delay operator. It is worth noting that all the entries in $\mathbf{H}^T(D)$ are monic polynomials, indicating that there exists at most 1 connection from each output symbol to any shift register. In Appendix B, we list some of our best RCDO codes using the D-transform representation in our study for reference. Eliminating the delay operator D, we

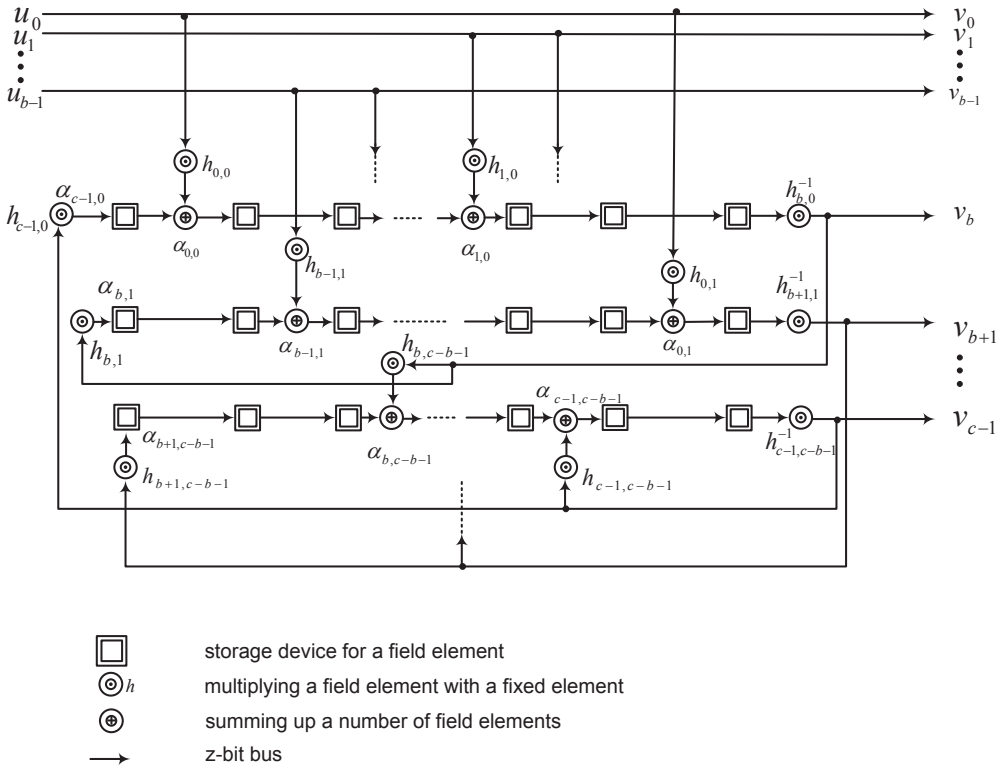


Figure 4.1 Encoder of rate $R = b/c$ q-ary RCDO codes.

obtain the matrix $\mathbf{H}^T$ representing the protograph of a q-ary RCDO codes :

$$\mathbf{H}^T = \begin{bmatrix} h_{0,0} & h_{0,1} & \dots & h_{0,c-b-1} \\ h_{1,0} & h_{1,1} & \dots & h_{1,c-b-1} \\ \vdots & \vdots & & \vdots \\ h_{c-1,0} & h_{c-1,1} & \dots & h_{c-1,c-b-1} \end{bmatrix}. \tag{4.4}$$

Note that $h_{i,j} \in GF(q)$, which distinguish the definition from that in Chapter 2 for the binary codes.

Fig. 4.2 is an example of a protograph with 6 variable nodes and 3 constraint nodes corresponding to the following parity-check matrix over $GF(8)$:

$$\mathbf{H}^T = \begin{bmatrix} 2 & 0 & 0 \\ 2 & 4 & 0 \\ 0 & 3 & 2 \\ 5 & 0 & 7 \\ 1 & 2 & 3 \\ 0 & 1 & 7 \end{bmatrix}. \tag{4.5}$$

In Fig. 4.2, each edge is labeled with its corresponding weight $h_{i,j}$ to facilitate the decoding

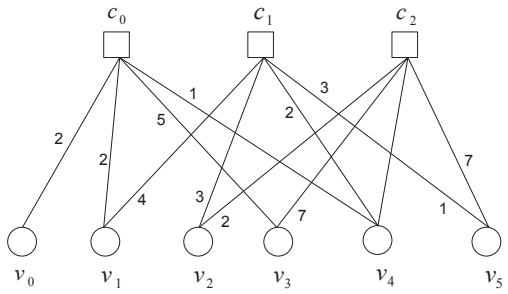


Figure 4.2 A protograph example with 6 variable nodes and 3 constraint nodes.

process. After selecting the delays $\{\alpha_{i,j}\}$, we obtain a matrix $\mathbf{H}^T(D)$:

$$\mathbf{H}^T(D) = \begin{bmatrix} 2D^4 & 0 & 0 \\ 2D & 4D^2 & 0 \\ 0 & 3D^2 & 2D^3 \\ 5 & 0 & 7D^2 \\ D^3 & 2 & 3D^3 \\ 0 & D & 7 \end{bmatrix}, \tag{4.6}$$

which specifies a rate 3/6 code with memory order $m = 4$, corresponding to the encoder in Fig. 4.3. Note that the the parity-check matrix in (4.6) does not necessarily represent an

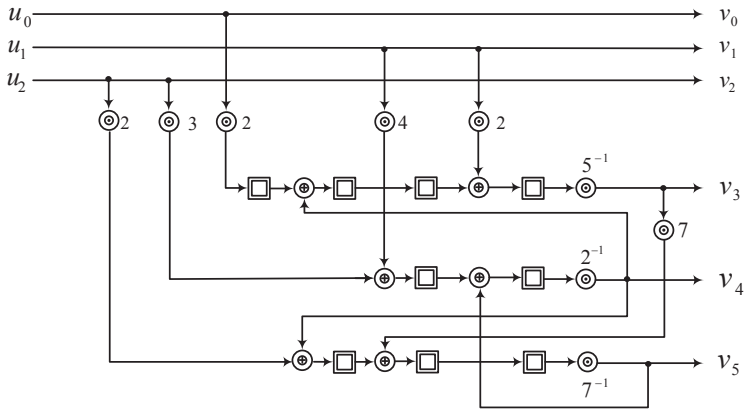


Figure 4.3 An encoder example for the rate 3/6 code represented by matrix $\mathbf{H}^T(D)$ of (4.6).

RCDO code, it is only used as an example of convolutional codes constructed with protographs.

As for the binary codes, the complete Tanner graph of a RCDO encoder can be obtained in two steps : i) duplicating a certain number of times the protograph associated with the encoder, ii) permuting the edges between all the duplicated protographs. The permutation of the edges is performed in accordance to the doubly orthogonal conditions, which define the RCDO codes. The doubly orthogonal conditions with this representation are described as in Section 2.1.1. To facilitate real implementations, in this thesis, we only consider RCDO codes with a regular Tanner graph, i.e., the connections to each of the variable nodes and the constraint nodes are given by a pair of constants (d_v, d_c). The doubly orthogonal conditions guarantee that the extended Tanner graphs of RCDO codes contains no cycles of lengths less than or equal to eight, helping the iterative decoding for this set of codes. Note that the protograph is duplicated m (the memory order) times before permutation is performed. In practice, it is always desirable to select the smallest m fulfilling the doubly orthogonal conditions in order to reduce the decoding latency.

4.2 Decoding of the q-ary CDO codes

In this section, we present the belief propagation algorithm applied in the decoding process for our RCDO codes. Similar to the case for non-recursive CDO codes in Chapter 3, the BP decoding for the RCDO codes is conducted using the FFT-based method ; however, in our later discussion on the effects of quantized messages in the BP decoding, the FFT-based scheme proves unsuitable for implementation with quantized message alphabets.

4.2.1 The FFT-based Belief Propagation decoding

Let $\lambda_{t,k}^{i,\mu}(\gamma_j), i = 0, 1, \ldots, c-1, j = 0, 1, \ldots, q-1$ represent the message transmitted by the variable node corresponding to the i^{th} received symbol at time t to the k^{th} properly delayed constraint node regarding the symbol γ_j during the μ^{th} iteration, i.e., it is the reliability for the event $v_t^i = \gamma_j$ given the information from all its incident constraint nodes except the k^{th} one. Note that here $k \in Y_i = \{n : h_{i,n} \neq 0\}$. Similarly, let $\varphi_{t,i}^{k,\mu}(\gamma_j), k = 0, 1, \ldots, c-b-1, j = 0, 1, \ldots, q-1$ denote the message transmitted by the k^{th} constraint node to the i^{th} variable node in the t^{th} duplicate of the protograph corresponding to the symbol γ_j during the μ^{th} iteration, i.e., it is the probability that the i^{th} parity-check equation is satisfied when $v_t^i = \gamma_j$ given the information from all except the i^{th} incident variable node, where $i \in Z_k = \{n : h_{n,k} \neq 0\}$.

The detailed decoding process consists of the following steps :

(a) Initialization

$$\lambda_{t,k}^{i,0}(\gamma_j) = f_t^i(\gamma_j), \tag{4.7}$$

where $i = 0, 1, \ldots, c-1, j = 0, 1, \ldots, q-1, k \in Y_i$. Here $f_t^i(\gamma_j)$ is the *a posteriori* probability for $v_t^i = \gamma_j$ calculated based on only the information received from the channel.

(b) Permutation and FFT

Define the vector of the messages sent from the i^{th} variable node to the k^{th} constraint node as

$$\boldsymbol{\lambda}_{t,k}^{i,\mu} = (\lambda_{t,k}^{i,\mu}(\gamma_0), \lambda_{t,k}^{i,\mu}(\gamma_1), \ldots, \lambda_{t,k}^{i,\mu}(\gamma_{q-1})). \tag{4.8}$$

$\boldsymbol{\xi}_{t,k}^{i,\mu} = (\xi_{t,k}^{i,\mu}(\gamma_0), \xi_{t,k}^{i,\mu}(\gamma_1), \ldots, \xi_{t,k}^{i,\mu}(\gamma_{q-1}))$ is defined to be the vector after the permutation and FFT operation on $\boldsymbol{\lambda}_{t,k}^{i,\mu}$, i.e.,

$$\boldsymbol{\xi}_{t-\alpha_{i,k},k}^{i,\mu} = \mathbf{FFT}(\mathbf{P}_{h_{i,k}}(\boldsymbol{\lambda}_{t-\alpha_{i,k},k}^{i,\mu})). \tag{4.9}$$

(c) Constraint node processing

The calculation at the constraint node is achieved with simple productions of the message vectors in the Fourier domain. We define the vector emitted by the k^{th} constraint node for the i^{th} variable node in the Fourier domain as

$$\boldsymbol{\theta}_{t,i}^{k,\mu} = (\theta_{t,i}^{k,\mu}(\gamma_0), \theta_{t,i}^{k,\mu}(\gamma_1), \ldots, \theta_{t,i}^{k,\mu}(\gamma_{q-1})); \tag{4.10}$$

therefore

$$\boldsymbol{\theta}^{k,\mu}_{t-\alpha_{i,k},i} = \prod_{j\in Z_k\backslash i} \otimes \boldsymbol{\xi}^{j,\mu}_{t-\alpha_{j,k},k}. \tag{4.11}$$

(d) Counter-permutation and iFFT

Denote $\boldsymbol{\varphi}^{k,\mu}_{t,i}$ as the message vector transmitted from the k th constraint node to the i th variable node, i.e.,

$$\boldsymbol{\varphi}^{k,\mu}_{t,i} = (\varphi^{k,\mu}_{t,i}(\gamma_0), \varphi^{k,\mu}_{t,i}(\gamma_1), \ldots, \varphi^{k,\mu}_{t,i}(\gamma_{q-1})). \tag{4.12}$$

It is calculated using

$$\boldsymbol{\varphi}^{k,\mu+1}_{t-\alpha_{i,k},i} = \mathbf{P}^{-1}_{h_{i,k}}(\mathbf{iFFT}(\boldsymbol{\theta}^{k,\mu}_{t-\alpha_{i,k},i})). \tag{4.13}$$

(e) Variable node processing

At a variable node, multiplications are applied to calculate the updated message in the probability domain to be transmitted to incident constraint nodes.

Define $\mathbf{f}^i_t = (f^i_t(\gamma_0), f^i_t(\gamma_1), \ldots, f^i_t(\gamma_{q-1}))$. The message vector $\boldsymbol{\lambda}^{i,\mu+1}_{t-\alpha_{i,k},k}$ is calculated using

$$\boldsymbol{\lambda}^{i,\mu+1}_{t-\alpha_{i,k},k} = \kappa \cdot \mathbf{f}^i_{t-\alpha_{i,k}} \otimes \prod_{j\in Y_i\backslash k} \otimes \boldsymbol{\varphi}^{j,\mu+1}_{t-\alpha_{i,j},i}. \tag{4.14}$$

Here κ is a normalization factor which guarantees that the probability for different choice of symbols for a single variable node sum up to 1, i.e., κ ensures

$$\sum_{j=0}^{q-1} \lambda^{i,\mu+1}_{t-\alpha_{i,k},k}(\gamma_j) = 1. \tag{4.15}$$

(f) Hard decision

If a variable node has been processed for N (a preset parameter) rounds of iterations, the reliability vector

$$\boldsymbol{\lambda}^i_t = (\lambda^i_t(\gamma_0), \lambda^i_t(\gamma_1), \ldots, \lambda^i_t(\gamma_{q-1}))$$

for the i th variable node is calculated with

$$\boldsymbol{\lambda}^i_{t-\alpha_{i,k}} = \mathbf{f}^i_{t-\alpha_{i,k}} \otimes \prod_{j\in Y_i} \otimes \boldsymbol{\varphi}^{j,N}_{t-\alpha_{i,j},i}. \tag{4.16}$$

The variable node is decoded into

$$v^i_t = \underset{\gamma_j}{\operatorname{argmax}}\, \lambda^i_t(\gamma_j). \tag{4.17}$$

It is also worth noting that the decoding process can be implemented using the pipeline structure as presented in (Jimenez Felstrom and Zigangirov, 1999).

4.2.2 Complexity

The numbers of different operations performed on a single edge in 1 iteration of the belief propagation decoding for q-ary RCDO codes are summarized in Table 4.1, with each of the edges carries the information, i.e., $\boldsymbol{\lambda} = (\lambda(\gamma_0), \lambda(\gamma_1), \ldots, \lambda(\gamma_{q-1}))$, for a z-bit symbol for $z = \log_2 q$. Note that all the operations are conducted in real arithmetics. The number of the most time consuming operations, i.e.,the multiplications, grows proportionally with q whilst the number of the less time-consuming additions grows as $O(qz)$. The number of operations in calculating the information for 1 bit on one edge are summarized and compared with those of binary RCDO decoding in the Log domain in Table 4.2, where we assume the decoding of the binary code follows the min-sum rule (Kschischang *et al.*, 2001). Note that d_v and d_c are the degrees of the variable nodes and the constraint nodes in the Tanner graph. In Table 4.2, the required multiplications are applied in the processing for constraint nodes and variable nodes, and the additions and table look-ups are needed to implement the FFT and permutation respectively. It is also possible to implement the multiplications in the Log domain, resulting in the Log-FFT domain decoding as suggested in (Song and Cruz, 2003). However, this method involves frequent transformations among probability, Log, and Fourier domains. In actual applications, the decoder is usually implemented with discrete message alphabets. Under this scenario, the additions and multiplications of the messages may be implemented easily with look-up tables (LUTs), consuming substantially less time than the arithmetic operations with real values.

Table 4.1 Number of operations needed for 1 iteration for 1 edge in BP decoding of q-ary RCDO codes ($q > 2$).

Additions	Multiplications	Table look-ups	Comparisons
$2qz$	$\frac{2q(d_c-2)}{d_c} + \frac{2q(d_v-2)}{d_v}$	$2q$	-

Table 4.2 Number of operations needed for 1 iteration for 1 bit on a single edge.

	Additions	Multiplications	Table look-ups	Comparisons
q-ary RCDO codes	$2q$	$\frac{2q(d_c-2)}{zd_c} + \frac{2q(d_v-2)}{zd_v}$	$\frac{2q}{z}$	-
Binary RCDO codes	$\frac{2(d_v-2)}{d_v}$	-	-	$d_c - 1$

4.3 Simulation results

This section presents computer simulation results for the q-ary RCDO codes under the BP decoding algorithm.

As in the case with the non-recursive codes, the computer simulations for the recursive codes have been carried out assuming binary phase shift keying (BPSK) modulation assuming an AWGN channel.

Throughout our simulation, we have used degree pairs (d_v, d_c) equal to $(3, 6)$. The simulations for q-ary RCDO codes are conducted for a set of rate 1/2 codes with various b, c and q values. The parameters $(b/c, q)$ are used to specify the codes in the following discussion. The parity-check matrices $\mathbf{H}^T(D)$ for different codes are obtained using those in (Roy *et al.*, 2010) by replacing those $h_{i,j}$ with value 1 in the binary matrix with non-zero elements in $GF(q)$. We select the weights $\{h_{i,j}\}$ uniformly randomly from non-zero elements in $GF(q)$. For each set of the results presented, we stop our simulations after 500 error bits had been observed.

The memory order m of the rate 1/2 RCDO codes under concern in the thesis are listed in Table 4.3. Examples of the q-ary RCDO codes in our simulation are provided in the Appendix, represented by their D-transform parity-check matrices $\mathbf{H}^T(D)$.

Table 4.3 Memory order of various codes. b/c : code rate ; m : memory order

b/c	4/8	5/10	6/12	8/16	9/18	10/20	15/30	30/60
m	33	33	34	40	48	49	149	150

It is suggested in (Davey and Markay, 1998) that the codes defined over $GF(q)$ with large values of q tend to enjoy good error performance due to the "locally dense" structure in their graphs. Therefore it could be expected that the benefit of large values for q mainly contributes mostly at relatively high E_b/N_0 region, when the cycles in the graph exert significant impacts on the error performance. The first set of simulations are conducted on codes of coding rate $R = b/c = 1/2$ with relatively small values for b. In the BP decoding process, although the decoder converges as the number of iterations increases, after a certain number of iterations, the calculated messages become inaccurate due to the existences of long cycles in the Tanner graph, potentially leading the decoder to converge to incorrect decisions. Therefore, it may not be beneficial to apply more than the sufficient number of iterations. The convention is to apply 50 iterations before the decoder determines the values of the variable nodes via hard decision ; after the 50^{th} iteration, the decoding becomes less efficient for most of the messages have already converged. Hence, we plot the BERs after the 50^{th} iteration for different values of q with $b = 4, 8$ are presented in Fig. 4.4. It is observed that with the same length and graph

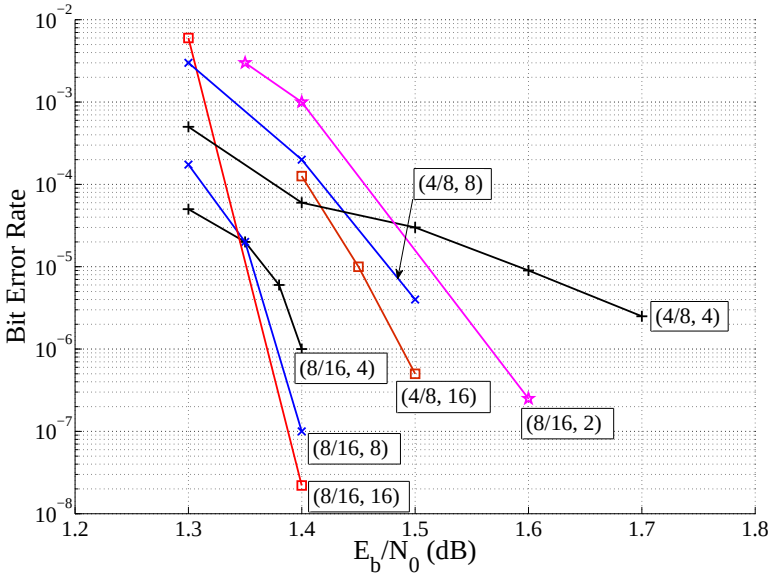


Figure 4.4 BER as a function of E_b/N_0 for various $(b/c, q)$ codes, with rate 4/8 and 8/16 codes and a binary code (Roy, 2011). $q = 4, 8, 16$; number of iterations = 50.

structure, although the codes with larger q values experienced higher bit error rates when E_b/N_0 is low, they tend to outperform their counterparts with smaller q values at higher E_b/N_0 region. For instance, for the rate 4/8 codes, doubling q have the effect of lowering the BERs by approximately 1 order of magnitude at $E_b/N_0 = 1.5$ dB. Similar benefit was observed by increasing the q value for the rate 8/16 codes, e.g., the $(8/16, 16)$ code achieves a bit error rate of 2×10^{-8} at $E_b/N_0 = 1.4$ dB while the $(8/16, 4)$ code demonstrates a BER of 10^{-6}. The performances of the rate 8/16 codes are also compared to their binary counterpart, whose BER curve for the binary codes is taken from (Roy, 2011). It is shown that in all the E_b/N_0 region of concern, our rate 8/16 q-ary RCDO codes with $q = 4, 8, 16$ achieve bit error rates several orders of magnitudes lower than the binary code.

The simulation results for rate 1/2 codes with larger b values are presented in Fig. 4.5. Similar to the codes with smaller protographs, it is shown that for the same constraint length, although codes with smaller q values demonstrate slightly earlier waterfall region, BERs for those with larger values of q catch up with them as E_b/N_0 increases, potentially providing lower error rates at higher E_b/N_0 region. For instance, for the rate 10/20 codes, the 4-ary code enjoys slightly better error rate than the 8-ary code in relatively low E_b/N_0 region ; however, the two curves crossed over at the bit error rate of 3×10^{-6}. For the rate 15/30 codes, the error performance of the 4-ary code surpasses those of the 8-ary code in the E_b/N_0 region under concern, although the bit error rate of the 8-ary code drops more quickly as E_b/N_0 gets large.

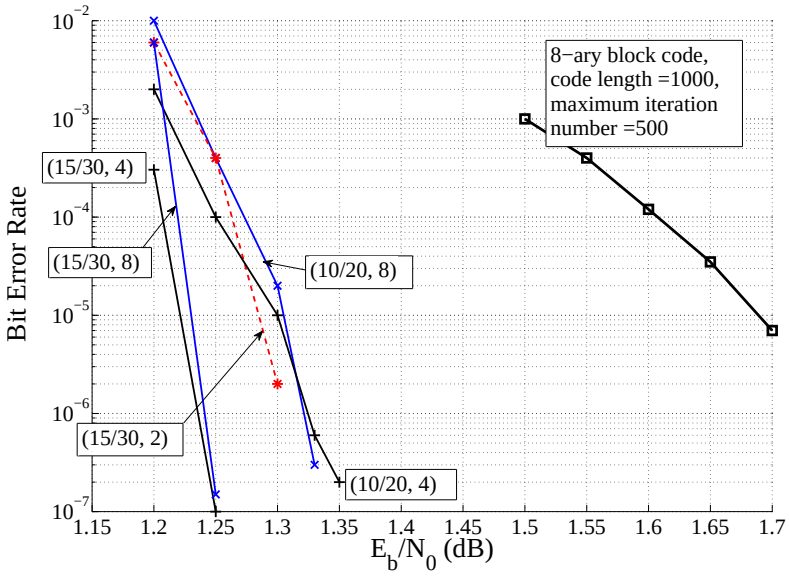


Figure 4.5 BER as a function of E_b/N_0 for rate 10/20 and 15/30 codes. $q = 2, 4, 8$; number of iterations = 50.

When E_b/N_0 is relatively low, the codes with large values for q are at a disadvantage because the large number of coordinates in a single message prevent the 'correct' coordinate from dominating the decoding; as E_b/N_0 gets large, the error performances for codes with large q improves quickly due to their denser structures, i.e., in the equivalent binary parity-check matrix for a single constraint node (as described in Chapter 2), each bit within a symbol is checked by $z = \log_2 q$ binary parity-check equations. However, due to limited simulation time, we were not able to compare their error performances for BER below 10^{-7}. We did not witness the error floors for our q-ary RCDO codes in the simulations; however, from our previous discussion, we suggest that the codes with larger values for q are preferred in terms of BER as E_b/N_0 further increases. Note that both of the codes enjoy better error performance than the binary code, especially at relatively higher E_b/N_0 region. In summary, the bit error rates of the codes with larger q values demonstrate the tendency to drop sharply as E_b/N_0 increases, whilst those with smaller q values are advantageous under lower E_b/N_0 region. The error performances of these RCDO codes are also compared to a rate 1/2 8-ary LDPC block code with code length 1000 and column weight 3, which has approximately the same constraint length with our $(10/20, 8)$ code, i.e., $(20 \times 45$ symbols). The BER curve of the block code is taken from (C.Davey, 1999); although the 8-ary block code had been decoded with a much larger number of iterations (maximum 500), the $(10/20, 8)$ code outperform the block code by more than 0.3 dB when BER $= 10^{-5}$.

As mentioned previously, the benefits on the error performances of the q-ary RCDO codes

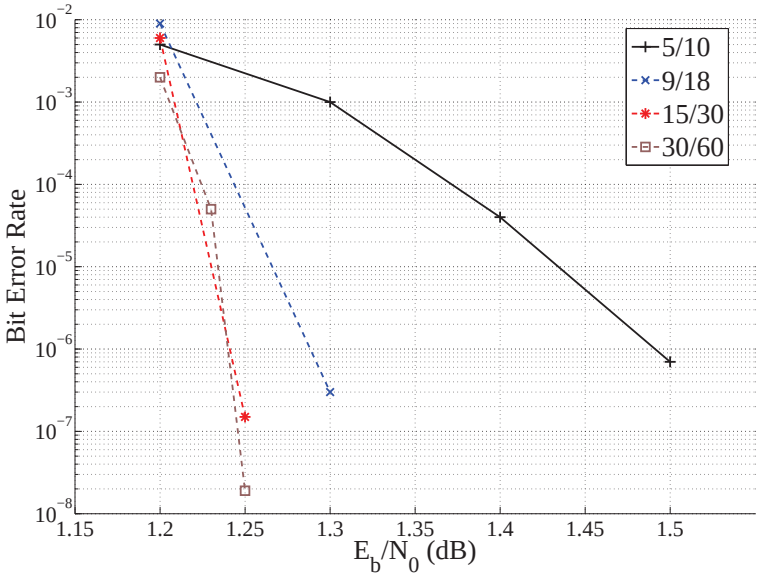


Figure 4.6 BER as a function of E_b/N_0 for rate 5/10 ; 9/18 ; 15/30, and 30/60 codes. $q = 8$; number of iterations =50

are mainly due to the relatively sparser connections in the Tanner graphs for these codes ; however, in Appendix A, it is demonstrated that the improvement may also be attributed to the increase in free distances of the q-ary RCDO codes as q increases.

Furthermore, the comparisons of error performances among rate 1/2 codes of the same alphabet size ($q = 8$) but with various b values are given in Fig. 4.6. For identical alphabet size, the codes with larger protographs demonstrate better error performances in the E_b/N_0 region under concern. For instance, both the $(15/30, 8)$ and the $(30/60, 8)$ codes outperform the $(5/10, 8)$ code by 0.25 dB at the BER of 10^{-6}. However, the gain by shifting to larger b values is gradually vanishing in the waterfall region as b increases, i.e., while a 0.2 dB gain is observed by moving from the rate 5/10 code to the rate 9/18 code, an improvement of only 0.05 dB is witnessed by further shifting to the rate 15/30 code. The benefit of using codes with larger b values is also observed when BER is low. For example, whilst the waterfall performances for the rate 15/30 and the rate 30/60 code are similar, the latter enjoys a bit error rate of 2×10^{-8} at $E_\mathrm{b}/N_0 = 1.25$ dB, or approximately 1 order of magnitude lower than that of the rate 15/30 code. Therefore, codes with larger protographs are desirable when extremely low bit error rate is required under harsh channel conditions.

Finally, we examine the effects of the number of iterations on the BER performances of the codes in Fig. 4.7. Unlike the case for the non-recursive CDO codes with threshold decoding (Cardinal *et al.*, 2003; Shen *et al.*, 2013), where further iterations after the 8 th round barely improves the error performances ; increasing the number of iterations continuously

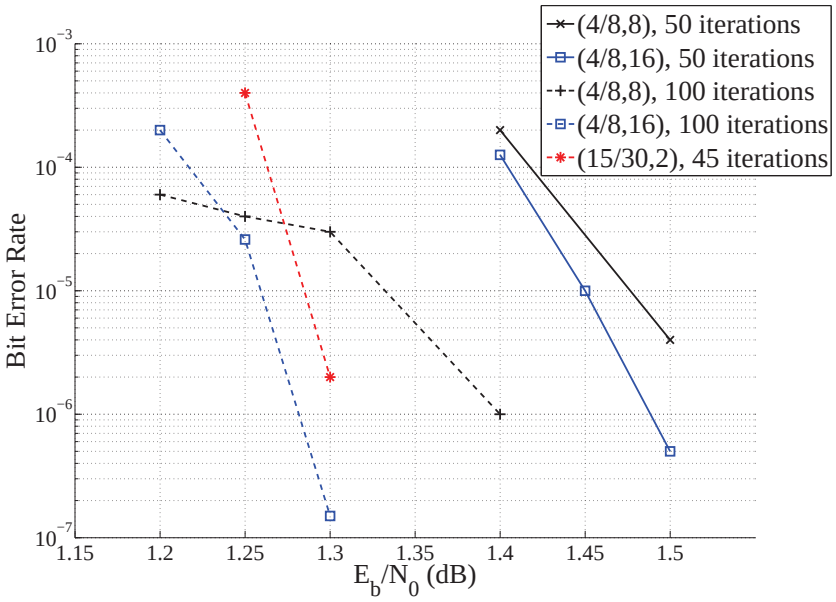


Figure 4.7 BER as a function of $E_{\rm b}/N_0$ for codes with parameters $(b/c, q)$; number of iterations =50, 100

improve the BER performances of the RCDO codes under the BP algorithm, which is similar to the binary CDO codes (He *et al.*, 2009). As shown in Fig. 4.7, the rate $(4/8, 16)$ code performs 0.2 dB better when it is decoded with 100 iterations rather than with 50 iterations. In addition, we again compare the performances of the rate 15/30 binary RCDO code with 45 iterations in (Roy *et al.*, 2010) with our codes in Fig. 4.7. It is observed that the $(4/8, 16)$ code with 100 decoding iterations outperforms the binary 15/30 code in the $E_{\rm b}/N_0$ region under concern. As shown Table 4.3, the memory order of the $(4/8, 16)$ code is 33 as compared to 149 for the binary 15/30 code. The decoding latency of the $(4/8, 16)$ code is therefore proportional to $8 \times 4 \times 33 \times 100 = 105600$ bits as compared to $30 \times 1 \times 149 \times 45 = 201150$ bits for the binary 15/30 code. Although we are applying more iterations (100 as opposed to 45), the latency of the $(4/8, 16)$ code is still reduced to around half as that of the binary 15/30 code. However, in terms of the decoding complexity, the $(4/8, 16)$ code is substantially more complex. In order to decode a single symbol, the information from d_v edges are required. As all the codes in our simulation have $d_v = 3$, we may conveniently compare the number of operations directly using those calculated in Table 3.1. In summary, it is calculated that the decoding process of the $(4/8, 16)$ code with 100 iterations is 15 times more complex than that of the binary rate 15/30 code with 45 iterations in the decoding, which is the cost for better error performance and short memory order (and therefore smaller latency). Note that although doubling the number of iterations improves the error performances of the q-ary RCDO codes as shown in

the above discussion ; we may be inclined to perform fewer iterations in order to reduce the overall complexity in the decoding process when the cost in error performances is tolerable when implementing the codes.

4.4 Quantized decoding of recursive CDO codes

This section addresses the quantization of the messages in the BP decoding for the q-ary RCDO codes. Although the FFT and Log-FFT algorithms reduce the number of operations in the constraint node processing, they are not suitable for quantized implementation due to numerical considerations. Hence, the quantized decoder is implemented using the Log-domain BP algorithm.

4.4.1 The quantized decoding scheme

In the FFT-based decoding algorithm, the messages computed are either probabilities or Fourier domain probabilities, i.e.,

$$\lambda_{t,k}^{i,\mu}(\gamma_j) \leq 1 \tag{4.18}$$

for $j = 0, 1, \ldots, q-1$; and due to the properties of FFT,

$$\xi_{t,k}^{i,\mu}(\gamma_j) \leq 2^{\frac{-z}{2}} \tag{4.19}$$

for $j = 0, 1, \ldots, q-1$, where $z = \log_2 q$.

Let g bits be used to quantize a single coordinate in a message, i.e., $\lambda_{t,k}^{i,\mu}(\gamma_j)$ and $\xi_{t,k}^{i,\mu}(\gamma_j)$, etc. The range $[0, 1]$ are divided into 2^g equal intervals, with the values in the range $[\frac{i-1}{2^g}, \frac{i}{2^g})$ represented by the integer i, for $i = 1, 2, \ldots, 2^g$. The multiplications in the FFT algorithm are performed using a pre-computed look-up table (LUT). However, due to the fact that most of the multiplicands are smaller than 1, the output for a single $\prod$ operation with a number of inputs is very likely to be a smaller value in the discrete message alphabet than all of the multiplicands, i.e., for the output

$$\text{Prob}(\prod = i) \gg \text{Prob}(\prod = j), \tag{4.20}$$

for $i < j$, where $i, j = 1, 2, \ldots, 2^g$. Therefore, even if the quantization applied to the messages is reasonably fine, the efficiency of the g-bit quantization diminishes over the decoding process for the large values in the message alphabet gradually lost their function. As a result, it is very often observed in our simulation that all coordinates in a message become identical in

the decoding process, i.e.,

$$Q(\lambda_{t,k}^{i,\mu}(\gamma_j)) = C, \tag{4.21}$$

for $j = 0, 1, \ldots, q-1$, where C is a constant and $Q(\cdot)$ represents the quantization function. Under this situation, the message implies that its corresponding variable node takes on the q elements in $GF(q)$ with equal probability, which is equivalent to the scenario when no message is provided for the variable node, i.e., the information contained in the message is lost. The problem can only be mitigated with very fine quantization applied for the small values.

Similar problems occur with the Log-FFT algorithm as described in (Song and Cruz, 2003), where the efficiency of the quantization diminishes with the frequent transformation among various message domains.

The Log-FFT algorithm replaces all the multiplicands in the FFT algorithm with their Log-domain counterparts, therefore reduces the probability that all the components in a single message become the same after a multiplicative operation. However, the quantized Log-FFT algorithm suffers from its own deficiency in the FFT process.

Each component of a message ranges from $-\infty$ to ∞. Since it is symmetrical around 0, we may quantize a finite subrange symmetrical around 0 uniformly, and leave $\pm\infty$ as the boundaries for our quantization. The range $(-\infty, \infty)$ is therefore separated into $(2^g - 1)$ intervals, with the quantization function $\mathscr{Q}(\cdot)$ defined as :

$$\mathscr{Q}(a) = \begin{cases} 2^{g-1} - 1, & \text{if } a > (2^{g-1} - \frac{1}{2})\delta \\ \lfloor \frac{a}{\delta} + \frac{1}{2} \rfloor \cdot \delta, & \text{if } \frac{\delta}{2} \leq a \leq (2^{g-1} - \frac{1}{2})\delta \\ -(2^{g-1} - 1), & \text{if } a < -(2^{g-1} - \frac{1}{2})\delta \\ \lceil \frac{a}{\delta} - \frac{1}{2} \rceil \cdot \delta, & \text{if } -(2^{g-1} - \frac{1}{2})\delta \leq a \leq -\frac{\delta}{2} \\ 0, & \text{otherwise} \end{cases}, \tag{4.22}$$

where δ is the quantization interval. This definition is similar to the quantization function in (Chung *et al.*, 2001a). As mentioned in (Song and Cruz, 2003), the operations in the FFT operations involves the calculation of $\log(\exp(u_1'') \pm \exp(u_2''))$, where u_1'' and u_2'' are 2 of the coordinates in a q-dimensional message. If one of u_1'' and u_2'' is large, $\log(\exp(u_1'') \pm \exp(u_2'')) = \max(u_1'', u_2'') + \mathscr{C}$, where $\mathscr{C}$ is a small value. With the quantized messages, it is likely that

$$\mathscr{Q}(\log(\exp(u_1'') \pm \exp(u_2''))) = \max(\mathscr{Q}(u_1''), \mathscr{Q}(u_2'')). \tag{4.23}$$

if $\mathscr{C}$ is smaller than the quantization interval δ. This means that after the FFT operation, the magnitude of all the coordinates in a single message vector are equal to the largest value among all the q coordinates before the FFT operation, i.e., the coordinates in a single

message take on an identical value. Hence, similar to the case with the FFT algorithm, the information contained in the message is lost, which is the observed phenomena in our tentative simulations. Since (4.23) is more likely when at least one of u_1'' and u_2'' is large, different from the case in the FFT algorithm, the Log-FFT algorithm requires finer quantization for the large values.

To avoid the above mentioned problem involved in the decoding process, it is then natural to apply the Log-domain BP decoding algorithm. Let the indices of the variable nodes incident to a single constraint node be denoted by $1, 2, \ldots, d_r$ for convenience. The received message on the i^{th} edge is therefore $\boldsymbol{\zeta}_i = (\zeta_i(\gamma_0), \zeta_i(\gamma_1), \ldots, \zeta_i(\gamma_{q-1}))$ which are the Log likelihood ratios (LLRs), where $\zeta_i(\gamma_j)$ is calculated using

$$\zeta_i(\gamma_j) = \log \frac{\lambda_{t,k}^{i,\mu}(\gamma_j)}{\lambda_{t,k}^{i,\mu}(\gamma_0)}. \tag{4.24}$$

Furthermore, denote $\boldsymbol{\varrho}_i = (\varrho_i(\gamma_0), \varrho_i(\gamma_1), \ldots, \varrho_i(\gamma_{q-1}))$ as the permutated version of $\boldsymbol{\zeta}_i$, i.e., $\boldsymbol{\varrho}_i = \mathbf{P}_{h_i}(\boldsymbol{\zeta}_i)$, where h_i is the weight on the i^{th} edge incident to the constraint node under discussion.

The processing in the constraint node follows the BCJR algorithm (Bahl *et al.*, 1974). Let

$$\mathbf{F}_i = (F_i(\gamma_0), F_i(\gamma_1), \ldots, F_i(\gamma_{q-1})) \tag{4.25}$$

denote the message for the sum from the 1 st to the i^{th} variable node incident on this constraint node, for $i = 1, 2, \ldots, d_r$. It is calculated using

$$\begin{aligned} \mathbf{F}_1 &= \boldsymbol{\varrho}_1, \\ F_i(\gamma_j) &= \log \sum_{\gamma_k + \gamma_l = \gamma_j} \exp(F_{i-1}(\gamma_k)) \cdot \exp(\varrho_i(\gamma_l)), \end{aligned} \tag{4.26}$$

for $i = 2, 3, \ldots, d_r, j = 0, 1, \ldots, q-1$. Similarly, let

$$\mathbf{B}_i = (B_i(\gamma_0), B_i(\gamma_1), \ldots, B_i(\gamma_{q-1})) \tag{4.27}$$

denote the message for the sum from the d_r^{th} to the i^{th} variable node, for $i = 1, 2, \ldots, d_r$. We have

$$\begin{aligned} \mathbf{B}_{d_r} &= \boldsymbol{\varrho}_{d_r}, \\ B_i(\gamma_j) &= \log \sum_{\gamma_k + \gamma_l = \gamma_j} \exp(B_{i+1}(\gamma_k)) \cdot \exp(\varrho_i(\gamma_l)), \end{aligned} \tag{4.28}$$

for $i = 1, 2, \ldots, d_r - 1, j = 0, 1, \ldots, q-1$. The message to be transmitted to the i^{th} variable node $\boldsymbol{\phi}_i = (\phi_i(\gamma_0), \phi_i(\gamma_1), \ldots, \phi_i(\gamma_{q-1}))$ is therefore calculated as

$$\begin{aligned} \boldsymbol{\phi}_1 &= \mathbf{B}_2, \\ \boldsymbol{\phi}_{d_r} &= \mathbf{F}_{d_r-1}, \\ \phi_i(\gamma_j) &= \log \sum_{\gamma_k+\gamma_l=\gamma_j} \exp(F_{i-1}(\gamma_k)) \cdot \exp(B_{i+1}(\gamma_l)), \end{aligned} \tag{4.29}$$

for $i = 2, 3, \ldots, d_r - 1, j = 0, 1, \ldots, q-1$.

The variable node updating for the LLRs are accordingly modified as

$$\boldsymbol{\zeta}^{i,\mu+1}_{t-\alpha_{i,k},k} = \boldsymbol{\zeta}^{i,0}_{t-\alpha_{i,k}} + \sum_{j \in Y_i \backslash k} \boldsymbol{\phi}^{j,\mu+1}_{t-\alpha_{i,j},i}. \tag{4.30}$$

Each coordinates of a message lies within $(-\infty, \infty)$, therefore we may apply the quantization function as specified in (4.22). Note that (4.22) differs with the quantization function in (Chung *et al.*, 2001a) in that (4.22) applies a finite number of bits, i.e., the message coordinates with absolute value larger than $(2^{g-1} - 1/2)\delta$ are determined to be $\pm(2^{g-1} - 1)$.

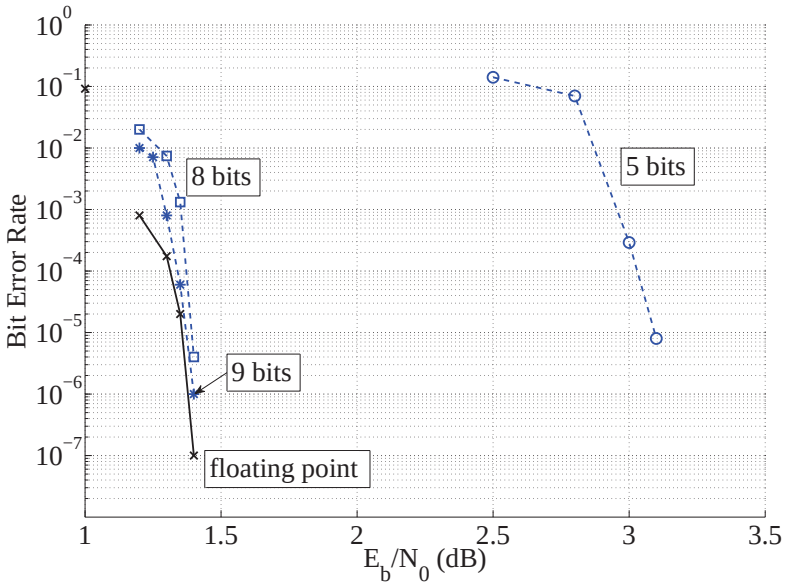
A further note on the weights

As for the q-ary non-recursive CDO codes, the weight selection affects the error performances of the recursive codes as well. In our discussion in Chapter 5, the influence of the message upon other messages in the same constraint node is explained using the BCJR algorithm as described above. Particularly, we examine (4.26) to explore how the messages from the i^{th} variable node affects that for the partial sum of the first i variable nodes; it will be shown that when the weights are randomly selected, the effect of incorrect messages from the i^{th} variable node is scattered in the calculation for $\mathbf{F}_i$.

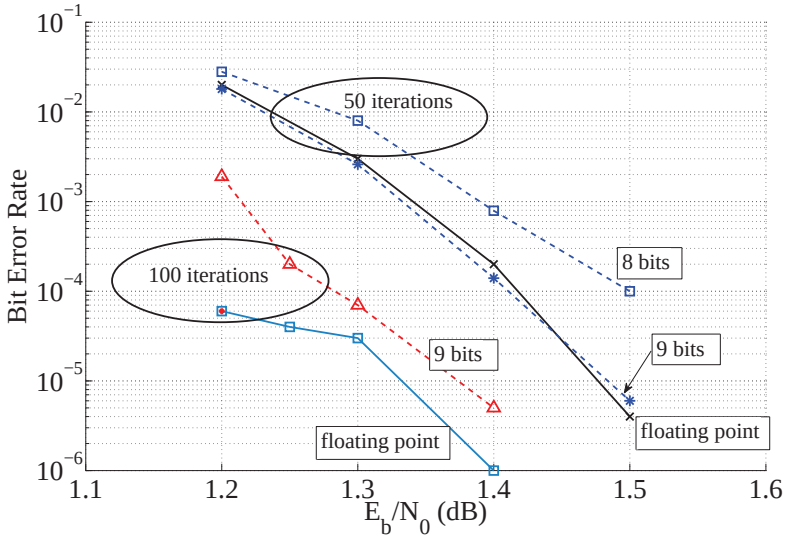
4.4.2 Error performances of the RCDO codes under the quantized decoder

This section provides the bit error performances obtained with computer simulations of the quantized decoder for some of the RCDO codes presented in Section 4.3; i.e., the codes with the parameters $(b/c, q)$ equal to $(4/8, 8)$, $(8/16, 8)$, $(15/30, 4)$, and $(15/30, 8)$.

The first set of simulations concerns the appropriate number of bits needed for the quantized decoder to achieve comparable error performance as floating number decoding. In our simulation, each coordinate in a message is quantized with g bits. Particularly, the quantiza-



(a) $(8/16, 8)$ code, number of iterations = 50



(b) $(4/8, 8)$ code, number of iterations = 50, 100

Figure 4.8 BER as a function of E_b/N_0 and the number of quantization bits for the $(8/16, 8)$ and $(4/8, 8)$ codes; number of iterations = 50, 100.

tion in (4.22) is applied with

$$\delta = \frac{\Delta}{2^{g-1}-1}, \tag{4.31}$$

i.e., the range $[-\Delta+\delta/2, \Delta-\delta/2]$ is uniformly separated with the interval δ; while the range $(-\infty, -\Delta+\delta/2)$ and $(\Delta-\delta/2, \infty)$ are represented by $-(2^{g-1}-1)$ and $(2^{g-1}-1)$ respectively. For a fixed g, the value of Δ affects the effectiveness of the quantized decoder. A larger value of Δ covers a wider range of the real valued LLRs, ameliorating the loss of information when message coordinates with absolute value greater than Δ have to be cut saturated; on the other hand, a large value of Δ indicates large quantization intervals (δ), rendering the computation less accurate when the LLRs are relatively small. Although optimal choice for Δ is not identical for different g; in our simulation, we found $\Delta = 50$ a reasonable value for most choices of g, taking into account both the range of messages covered and the quantization precision.

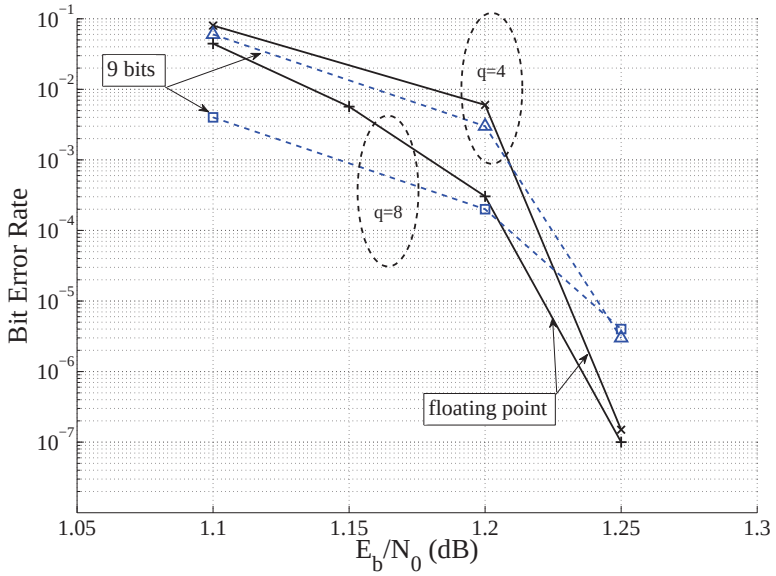


Figure 4.9 BER as a function of E_b/N_0 for rate 15/30 code. $q = 4, 8$; number of iterations = 50.

The curves in Fig. 4.8 compare the BER performances of the quantized decoder for the $(8/16, 8)$ and the $(4/8, 8)$ codes with various number of quantization bits g with that of floating point decoder after the 50 th and the 100 th iteration. It is observed that the value of g considerably affects the performance of the quantized decoder. For the $(8/16, 8)$ code, while applying 8-bit quantization has a large improvement (more than 1.5 dB) over the case with 5-bit quantization, 9-bit quantization scheme offers little advantage over that with 8 bits. Although as shown in Fig. 4.8(a), 9-bit scheme offers error performances close to the floating

number scheme. Hence it is concluded that $g = 9$ is an appropriate choice compromising complexity with performance for the $(8/16, 8)$ code after 50 th iteration. Similarly, for the $(4/8, 8)$ code after the 50 th iteration, 9 bits for each coordinates in a message is sufficient for the decoder to achieve similar BER performances to the floating point decoding. However, as shown in Fig. 4.8(b), increasing the number of iterations enlarge the performance gap between the quantized decoder and the floating point decoder; i.e., for the $(4/8, 8)$ code, although the 9-bit quantization achieves almost the same error performances as the floating point decoding when 50 iterations are conducted, discernible difference is witnessed between the two when the number of iterations is increased to 100. With a larger number of iterations, the messages tend to converge to a certain smaller range of values; therefore in order to achieve error performances close to the floating point decoder, finer quantization is required to distinguish between message values that are close to each other. Finally, we investigate the error performances of codes with relatively large b values under the quantized decoding, as shown in Fig. 4.9. The performance gap between the quantized decoder and the floating point decoder is slightly enlarged for the rate 15/30 codes; i.e., at $E_{\mathrm{b}}/N_0 = 1.25$ dB, the 9-bit quantized decoder demonstrates BERs more than 1 order of magnitude higher than the floating point decoder for both the 4-ary and 8-ary codes.

In conclusion, with sufficient number of quantization bits, the quantized decoding scheme is observed to provide error performances very close to floating point decoding when E_{b}/N_0 is low.

4.5 Summary

In this chapter, we adapted the binary recursive convolutional doubly orthogonal codes to the use of finite fields $GF(q)$ as alphabets. The belief propagation algorithm for this new set of codes is described in detail; the implementation complexity of the BP algorithm for this set of codes is calculated in terms of the numbers of different arithmetic operations. Simulation results demonstrate that increasing the alphabet size q leads to lower bit error rate in relatively high E_{b}/N_0 region, while the BERs of the codes with smaller q values drops slightly earlier in lower E_{b}/N_0 region. Moreover, it is shown that q-ary RCDO codes are advantageous over the q-ary LDPC codes with comparable lengths in terms of error performances. In our simulations, it is also demonstrated that q-ary RCDO codes of relatively small protographs with large value of q may achieve better error performances than longer binary RCDO codes; the benefit of which is the smaller latency in terms of the length of the bit streams stored in the pipelined decoder, owing to the smaller memory order of these codes. However, these benefits are obtained at the price of increased decoding complexity;

in practice, the q-ary RCDO codes provide an alternative in the tradeoff between decoding complexity and error performances. In summary, the q-ary RCDO codes help to improve the bit error rate at higher E_{b}/N_0 regions, as well as to decrease the latency in the decoding process. The quantized decoding scheme for the proposed set of codes are considered with the Log-domain BP decoding algorithm to facilitate the real applications. It is demonstrated through simulations that applying a certain number ($g = 9$) of bits in quantizing the message alphabet would be quite sufficient for the decoder to achieve comparable error performances as the floating number decoder; whilst still more bits lead to little improvement in the BERs. However, finer quantization may be required if a large number (100) of iterations is applied to accommodate the converged message in later iterations, leading to yet increased implementation complexity.

CHAPTER 5

Code Determination : Uniformly Random Weights against Identical Weights

In our previous discussion on the non-recursive and recursive q-ary CDO codes, we assigned uniformly randomly selected weights to the edges in the corresponding Tanner graphs of our q-ary CDO codes. In Chapter 3 and Chapter 4, we briefly mentioned the origin of the differences in error performances between codes with randomly selected weights and identical weights ; this chapter provides some detailed discussion of the effects of weight selection on the error performances of the q-ary CDO codes under the iterative decoding algorithms.

Simulation results demonstrate that in terms of the error performances of these codes, the random selection is advantageous over assigning identical weights for all the edges. It is illustrated that with identical weights, erroneous messages affect the decoding process more easily in the short-length cycles via the constraint node updating of these codes. It is also demonstrated that the q-ary CDO codes with identically selected weights could be linked to the binary CDO codes ; through the evaluation of decoding thresholds for this particular set of codes, further insight is gained on the influence of weight selection for our q-ary CDO codes. Note that although the simulation results presented in this chapter focus on the q-ary CDO codes with single-shift-register structures, the analysis applies to other q-ary codes decoded with iterative decoding algorithm as well.

5.1 Performances of identical weights against random weights selection

In this section, we examine through simulation the effects of the selection of weights in the Tanner graphs on the error performances of the q-ary CDO codes. In particular, we compare the error performances of the q-ary non-recursive CDO codes with random weights as presented in Chapter 3 against those with all their weights equal to 1.

In Fig. 5.1, the error performances of q-ary CDO codes with weights $\{w_i\}$ uniformly randomly selected from the non-zero elements in $GF(q)$ are compared to those with identical weights, i.e., $w_i = 1$ for all i. It is observed that the codes with identical weights suffer from higher error floors. For instance, when $J = 10, q = 8$, although the code with identical weights performs better when E_b/N_0 is relatively low, when E_b/N_0 increases to 3.3 dB, the code with randomly selected weights enjoys a BER approximately two orders of magnitudes lower than that of the code with identical weights.

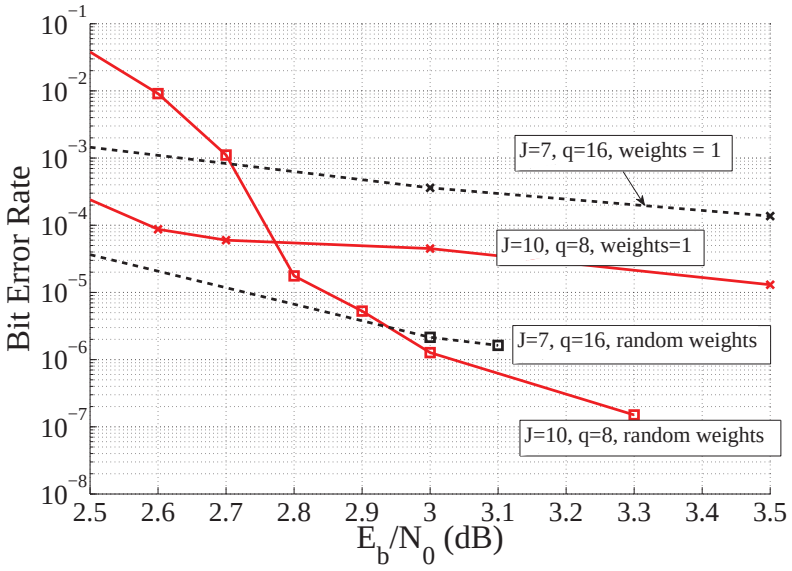


Figure 5.1 BER as a function of E_b/N_0 for CDO codes with random or identical weights, 8 iterations.

5.2 Analysis on the effects of weight selection

The advantage of the randomly selected weights over identical weights has its roots in the message propagation process in short-length cycles through the constraint node updating. Furthermore, the q-ary codes with identical weighs are further shown to be related to the the binary codes, eliminating the merits of the locally denser structures of the parity-check equations over $GF(q)$.

5.2.1 From message propagation perspective

The disadvantage of the codes with identically selected weights in terms of error performances in higher E_b/N_0 regions could be attributed to the enhancement of incorrect messages in short-length cycles in the corresponding Tanner graphs. With $\{w_i\} = \{1\}$, the parity-check equations are reduced to simple summations over the involved variable nodes. Denote v_i and v_j as 2 variable nodes involved in a cycle. In the decoding process, once v_i converges to an incorrect choice of symbol, all the other involved variable nodes are affected by this message through various parity-check equations. Since v_j is treated identically in different parity-check equations, the erroneous message from v_i is enhanced at v_j, forcing v_j to converge to an incorrect choice of symbol through different constraint nodes.

For the convenience of discussion, we illustrate the above point with simplified examples. Two simple examples of length-4 and length-6 cycles are given as follows. Please note that

there exist no cycles of length-4 or 6 in the CDO codes, but the underlying principle remains the same for cycles of larger lengths.

Table 5.1 Parity-check equations for a length-4 cycle

	$w_i = 1$	w_i is r.v.
c_1	$v_1 + v_2 = 0$	$v_1 + v_2 = 0$
c_2	$v_1 + v_2 = 0$	$v_1 + 2v_2 = 0$

Consider a length-4 cycle consisting of variable nodes $v_i, i = 1, 2$ and constraint nodes $c_i, i = 1, 2$. Assume that the code is defined in $GF(4)$ and the sets of parity-check equations with identical weights and random weights are given in Table 5.1.

Table 5.2 Parity-check equations for a length-6 cycle

	$w_i = 1$	w_i is r.v.
c_1	$v_1 + v_2 = 0$	$v_1 + v_2 = 0$
c_2	$v_1 + v_2 = 0$	$v_1 + 2v_2 = 0$
c_3	$v_1 + v_2 = 0$	$v_1 + 3v_2 = 0$

Suppose the transmitted symbols are $v_1 = v_2 = 0$, and in the decoding, the message for v_1 incorrectly converges at $v_1 = 3$. For the code with identical weights, the parity-check equations for c_1 and c_2 both force v_2 to converge at $v_2 = 3$, thus the erroneous message from v_1 is enhanced in this cycle through the two parity-check equations; therefore v_2 is likely to be decoded as 3. For randomly selected weights, c_1 forces v_2 to converge at $v_2 = 3$, whilst c_2 forces it to converge at $v_2 = 3 \cdot 2^{-1} = 2$; scattering the incorrect message on different choices for v_2, which is less harmful compared with the case with all identical weights.

For a length-6 cycle with variable nodes $v_i, i = 1, 2, 3$ and constraint nodes $c_i, i = 1, 2, 3$. Suppose the parity-check equations with the identical weights and random weights are given in Table 5.2.

Similarly, let us assume that transmitted symbols are $v_1 = v_2 = v_3 = 0$, and v_1 is mistakenly decoded as $v_1 = 3$. In the code with identical weights, constraint node c_1 forces that $v_2 = 3$, c_2 forces that $v_3 = 3$ and thus c_3 forces v_2 into 3. Therefore, the incorrect message is enhanced at v_2 for both of its incident constraint nodes (c_1 and c_3) tend to push the decision $v_2 = 3$. In the code with random weights, c_1 forces that $v_2 = 3$, c_2 forces that $v_3 = 3 \cdot 2^{-1}$ and c_3 forces v_2 into $3 \cdot 2^{-1} \cdot 3 = 1$. Since the incident constraint nodes of v_2 prefer different choices of v_2 (3 and 1), the effect of the erroneous message from v_1 is again scattered, leading to less detrimental effect than in the code with identical weights.

In summary, we may conclude that it is desirable to optimize the selection of weights for every single cycle in the Tanner graph of a CDO code by making the decision on one variable node, through all the involved constraint nodes, correspond to all different decisions on another variable node in the same cycle. However, a solution of this scheme may not be achievable for large number of iterations, for the number of cycles continues to increase with the iteration number and the choices of weight for each connection is limited ($(q-1)$ possibilities). From the results of our simulations, the uniformly random selection of the weights is a reasonably good method.

In the following, we discuss the means through which an erroneous message affects other variable nodes connected to the same constraint nodes.

5.2.2 Influences of weights on the erroneous messages in decoding

In the previous section, it is shown that a variable node with erroneous message 'forces' other variable nodes connected to the same constraint node to converge to wrong values. In this section, we provide some insight into the interaction among messages involved in the same parity-check equation; particularly, we discuss how an erroneous message affects the message updating process.

It is most convenient to consider the Log-domain decoding algorithm as described in Chapter 4, where the constraint node updating process follows the BCJR algorithm (Bahl *et al.*, 1974). In the following analysis, we assume that the transmitted symbols are all γ_0 (0). It is obvious that the message transmitted to a constraint node $\boldsymbol{\zeta}_i$ is correct iff

$$\underset{\gamma_j}{\operatorname{argmax}}\, \zeta_i(\gamma_j) = \gamma_0, \tag{5.1}$$

where $j = 0, 1, \ldots, q-1$. Similarly, we may define the message $\mathbf{F}_i$ on the partial sum from the 1 st to the $i^{\,th}$ variable node to be *correct* if

$$\underset{\gamma_j}{\operatorname{argmax}}\, F_i(\gamma_j) = \gamma_0. \tag{5.2}$$

This is because when the messages of the variable nodes involved in $\mathbf{F}_i$ are all correct, their corresponding symbols sum to γ_0.

For a message $\mathbf{a} = (a(\gamma_0), a(\gamma_1), \ldots, a(\gamma_{q-1}))$, denote the event that $\mathbf{a}$ is correct as $\mathbf{a}^c$ and the event that $\mathbf{a}$ is erroneous as $\mathbf{a}^e$. The message $\mathbf{F}_i$ is calculated using $\mathbf{F}_{i-1}$ and $\boldsymbol{\varrho}_i$ as in (4.26). The probability that $\mathbf{F}_i$ is erroneous is

$$\text{Prob}(\mathbf{F}_i^e) \;=\; \text{Prob}(\mathbf{F}_{i-1}^c)\text{Prob}(\mathbf{F}_i^e|\mathbf{F}_{i-1}^c) + \text{Prob}(\mathbf{F}_{i-1}^e)\text{Prob}(\mathbf{F}_i^e|\mathbf{F}_{i-1}^e)$$

$$\geq \quad \text{Prob}(\mathbf{F}_{i-1}^{c})\text{Prob}(\mathbf{F}_{i}^{e}|\mathbf{F}_{i-1}^{c}). \tag{5.3}$$

Furthermore, denote $\mathbf{a}^{\rightarrow\gamma_i}$ as the event that the message $\mathbf{a}$ is erroneous and is mistaken into γ_i, i.e.,

$$\text{Prob}(\mathbf{a}^{\rightarrow\gamma_i}) = \text{Prob}(\underset{\gamma_j}{\text{argmax}}\, a(\gamma_j) = \gamma_i), \tag{5.4}$$

for $i \neq 0$. Obviously, under the assumption that all the transmitted symbols are γ_0,

$$\sum_{i=1}^{q-1} \text{Prob}(\mathbf{a}^{\rightarrow\gamma_i}) = \text{Prob}(\mathbf{a}^{e}). \tag{5.5}$$

Therefore,

$$\begin{aligned} \text{Prob}(\mathbf{F}_{i}^{e}|\mathbf{F}_{i-1}^{c}) &= \sum_{j=1}^{q-1} \text{Prob}(\mathbf{F}_{i}^{\rightarrow\gamma_i}|\mathbf{F}_{i-1}^{c}) \\ &= \sum_{j=1}^{q-1} \text{Prob}(\underset{\gamma_k}{\text{argmax}}\, F_i(\gamma_k) = \gamma_j|\mathbf{F}_{i-1}^{c}) \end{aligned} \tag{5.6}$$

First consider the case when all the weights are identical. Without loss of generality, let us assume that all the weights are 1, therefore

$$\text{Prob}(\mathbf{F}_{i}^{e}|\mathbf{F}_{i-1}^{c}) = \sum_{j=1}^{q-1} \text{Prob}([\log \sum_{\gamma_k+\gamma_l=\gamma_j} \exp(F_{i-1}(\gamma_k))\cdot\exp(\zeta_i(\gamma_l))] > F_i(\gamma_h \neq \gamma_j)|\mathbf{F}_{i-1}^{c}). \tag{5.7}$$

Since $\exp(F_{i-1}(\gamma_k))\cdot\exp(\zeta_i(\gamma_l)) \geq 0$ for all i, k, l, and $F_{i-1}(\gamma_0)$ is the most dominant coordinate in $\mathbf{F}_{i-1}$ when $\mathbf{F}_{i-1}$ is correct, (5.7) is further bounded by the following :

$$\begin{aligned} \text{Prob}(\mathbf{F}_{i}^{e}|\mathbf{F}_{i-1}^{c}) &\geq \sum_{j=1}^{q-1} \text{Prob}([\log(\exp(F_{i-1}(\gamma_0)) \cdot \exp(\zeta_i(\gamma_j)))] > F_i(\gamma_h \neq \gamma_j)|\mathbf{F}_{i-1}^{c}), \\ &= \sum_{j=1}^{q-1} \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j)] > F_i(\gamma_h \neq \gamma_j)|\mathbf{F}_{i-1}^{c}) \\ &= \sum_{j=1}^{q-1}\sum_{k=0}^{q-1} \text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_k}|\mathbf{F}_{i-1}^{c})\text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j)] \\ &\qquad\qquad > F_i(\gamma_h \neq \gamma_j)|\boldsymbol{\zeta}_i^{\rightarrow\gamma_k}, \mathbf{F}_{i-1}^{c}) \\ &= \sum_{j=1}^{q-1}\sum_{k=0}^{q-1} \text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_k})\text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j)] > F_i(\gamma_h \neq \gamma_j)|\boldsymbol{\zeta}_i^{\rightarrow\gamma_k}, \mathbf{F}_{i-1}^{c}) \end{aligned}$$

$$\begin{aligned} &\geq \sum_{j=1}^{q-1} \text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_j}) \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j)] > F_i(\gamma_h \neq \gamma_j) | \boldsymbol{\zeta}_i^{\rightarrow\gamma_j}, \mathbf{F}_{i-1}^c) \\ &\triangleq \sum_{j=1}^{q-1} \text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_j}) \cdot S_j. \end{aligned} \tag{5.8}$$

Note that when

$$\begin{aligned} F_{i-1}(\gamma_0) &\gg F_{i-1}(\gamma_k) \text{ for } k \neq 0, \\ \zeta_i(\gamma_j) &\gg \zeta_i(\gamma_k) \text{ for } k \neq j, \end{aligned}$$

The inequalities in (5.8) could be replaced with equalities.

Therefore, we have

$$\text{Prob}(\mathbf{F}_i^e) \geq \text{Prob}(\mathbf{F}_{i-1}^c) \sum_{j=1}^{q-1} \text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_j}) \cdot S_j. \tag{5.9}$$

Equivalently,

$$\text{Prob}(\mathbf{F}_i^c) \leq 1 - \text{Prob}(\mathbf{F}_{i-1}^c) \sum_{j=1}^{q-1} \text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_j}) \cdot S_j \triangleq C_1. \tag{5.10}$$

We now consider using randomly selected weights. For the case of randomly selected weights, denote the weight on the $i^{\,th}$ edge connected to the constraint node by w_i. Similar to the previous derivations, (5.7) now becomes

$$\begin{aligned} \text{Prob}(\mathbf{F}_i^e | \mathbf{F}_{i-1}^c) &= \sum_{j=1}^{q-1} \text{Prob}([\log \sum_{\gamma_k + w_i \gamma_l = \gamma_j} \exp(F_{i-1}(\gamma_k)) \cdot \exp(\zeta_i(\gamma_l))] > F_i(\gamma_h \neq \gamma_j) | \mathbf{F}_{i-1}^c) \\ &\geq \sum_{j=1}^{q-1} \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot w_i^{-1})] > F_i(\gamma_h \neq \gamma_j) | \mathbf{F}_{i-1}^c) \\ &= \sum_{j=1}^{q-1} \sum_{k=0}^{q-1} \text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_k} | \mathbf{F}_{i-1}^c) \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot w_i^{-1})] \\ &\qquad > F_i(\gamma_h \neq \gamma_j) | \boldsymbol{\zeta}_i^{\rightarrow\gamma_k}, \mathbf{F}_{i-1}^c) \\ &\geq \sum_{j=1}^{q-1} \text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_j}) \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot w_i^{-1})] > F_i(\gamma_h \neq \gamma_j) | \boldsymbol{\zeta}_i^{\rightarrow\gamma_j}, \mathbf{F}_{i-1}^c), \end{aligned} \tag{5.11}$$

for $h \neq j$. Each summand in the last line of (5.11) is the probability that $\mathbf{F}_i$ is in error when the $i^{\,th}$ variable node is mistaken into γ_j. However, it is different from (5.8) in that the

greatest coordinate in $\mathbf{F}_i$ is still dependent on the weight w_i; i.e., when the i^{th} variable node is mistaken into $\gamma_j \neq \gamma_0$, most likely $\mathbf{F}_i$ is mistaken into $\gamma_j \cdot w_i^{-1}$.

For randomly selected w_i,

$$\begin{aligned} & \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot w_i^{-1})] > F_i(\gamma_h \neq \gamma_j) | \boldsymbol{\zeta}_i^{\rightarrow \gamma_j}, \mathbf{F}_{i-1}^c) \\ = & \sum_{w_i \neq \gamma_0} \text{Prob}(w_i) \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot w_i^{-1})] > F_i(\gamma_h \neq \gamma_j) | \boldsymbol{\zeta}_i^{\rightarrow \gamma_j}, \mathbf{F}_{i-1}^c, w_i) \\ = & \frac{1}{q-1} \sum_{w_i \neq \gamma_0} \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot w_i^{-1})] F_i(\gamma_h \neq \gamma_j) | \boldsymbol{\zeta}_i^{\rightarrow \gamma_j}, \mathbf{F}_{i-1}^c, w_i). \end{aligned} \tag{5.12}$$

Note that

$$\begin{aligned} \underset{\gamma_k}{\text{argmax}} F_{i-1}(\gamma_k) &= \gamma_0 \\ \underset{\gamma_k}{\text{argmax}} \zeta_i(\gamma_k) &= \gamma_j, \end{aligned} \tag{5.13}$$

When $w_i \neq 1$, it is obvious that

$$\text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot w_i^{-1})] > F_i(\gamma_h \neq \gamma_j) | \boldsymbol{\zeta}_i^{\rightarrow \gamma_j}, \mathbf{F}_{i-1}^c, w_i) = 0. \tag{5.14}$$

This is because

$$F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot w_i^{-1}) < F_{i-1}(\gamma_0) + \zeta_i(\gamma_j) < F_i(\gamma_h = \gamma_j \cdot w_i). \tag{5.15}$$

Therefore,

$$\begin{aligned} & \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot w_i^{-1})] > F_i(\gamma_h \neq \gamma_j) | \boldsymbol{\zeta}_i^{\rightarrow \gamma_j}, \mathbf{F}_{i-1}^c) \\ = & \frac{1}{q-1} \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot 1^{-1})] F_i(\gamma_h \neq \gamma_j) | \boldsymbol{\zeta}_i^{\rightarrow \gamma_j}, \mathbf{F}_{i-1}^c, 1) \\ = & \frac{1}{q-1} \cdot S_j. \end{aligned} \tag{5.16}$$

That is to say, when the weight w_i is randomly selected, the probability that $\mathbf{F}_i$ is mistaken into γ_i when ζ_i is mistaken into γ_i is reduced, which is intuitive for $\mathbf{F}_i$ is the message involving just one more variable node (c_i) as compared to $\mathbf{F}_{i-1}$.

Combining (5.16) and (5.11), we have

$$\text{Prob}(\mathbf{F}_i^e | \mathbf{F}_{i-1}^c) \geq \sum_{j=1}^{q-1} \text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow \gamma_j}) \text{Prob}([F_{i-1}(\gamma_0) + \zeta_i(\gamma_j \cdot w_i^{-1})] > F_i(\gamma_h \neq \gamma_j) | \boldsymbol{\zeta}_i^{\rightarrow \gamma_j}, \mathbf{F}_{i-1}^c)$$

$$= \frac{1}{q-1}\sum_{j=1}^{q-1}\text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_j})\cdot S_j. \tag{5.17}$$

Therefore,

$$\text{Prob}(\mathbf{F}_i^e) \geq \frac{1}{q-1}\text{Prob}(\mathbf{F}_{i-1}^c)\sum_{j=1}^{q-1}\text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_j})\cdot S_j. \tag{5.18}$$

Equivalently,

$$\text{Prob}(\mathbf{F}_i^c) \leq 1 - \frac{1}{q-1}\text{Prob}(\mathbf{F}_{i-1}^c)\sum_{j=1}^{q-1}\text{Prob}(\boldsymbol{\zeta}_i^{\rightarrow\gamma_j})\cdot S_j \triangleq C_h. \tag{5.19}$$

Comparing (5.10) and (5.19), the probability that F_i is correct is upper bounded by a function of the error probabilities of $\mathbf{F}_{i-1}$ and $\boldsymbol{\zeta}_i$. C_h is greater than C_1 due to the factor $1/(q-1)$ before the subtrahend. Therefore, the random selection of weights is advantageous over the identical selection for it scatters the effects of an erroneous $\boldsymbol{\zeta}_i$ over the $(q-1)$ coordinates of $\mathbf{F}_i$.

Note that in the above discussion on the error probabilities, we have used a few inequalities to simplify the calculation. These inequalities make the derivation less strict ; however, the principle on the influence of erroneous messages among variable nodes in the same parity-check equation remains the same : randomly selected weights scatters the influence of erroneous messages onto different coordinates of neighboring variable nodes in the same constraint node.

A similar discussion can be applied to the error probabilities of $\mathbf{B}_i$.

As a conclusion, it is preferable to select the weights in the q-ary codes randomly rather than to select them with identical values.

5.2.3 Codes with identical weights and the binary codes

In Chapter 2, it is demonstrated that each element in $GF(q = 2^z)$ can be represented by a $z \times z$ binary matrix. A single constraint node is defined by the parity-check equation

$$\sum_{i=1}^{d_r} c_i h_i = 0, \tag{5.20}$$

where $c_i \in GF(q)$ represents the $i^{\,th}$ variable node incident of the constraint node under concern. Denote the binary image of h_i by the square matrix $\mathbf{H}_i$ and the symbol c_i by a

binary vector $\mathbf{b}_i = (b_{i,1}, b_{i,2}, \cdots, b_{i,z})$, the parity-check equation in is rewritten as

$$\sum_{i=1}^{d_r} \mathbf{b}_i \mathbf{H}_i = \mathbf{0}, \tag{5.21}$$

which defines a binary parity-check code with parity-check matrix

$$\mathbf{S} = \begin{bmatrix} \mathbf{H}_1^T & \mathbf{H}_2^T & \dots & \mathbf{H}_{d_r}^T \end{bmatrix}, \tag{5.22}$$

and code block $\mathbf{c} = (\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_{d_r})$.

The binary parity-check matrix is determined through a vector of weights $(h_1, h_2, \dots, h_{d_r})$. A special case is when $h_i = 1, i = 1, 2, \dots, d_r$. The binary image for 1 is a $z \times z$ binary identity matrix. Therefore, the parity-check matrix takes the form as follows :

$$\mathbf{S} = \begin{bmatrix} 1 & & & 1 & & & \dots & 1 & & \\ & 1 & & & 1 & & \dots & & 1 & \\ & & \ddots & & & \ddots & \dots & & & \ddots \\ & & & 1 & & & 1 & \dots & & & 1 \end{bmatrix}. \tag{5.23}$$

Apparently, within the code block $\mathbf{c}$, $b_{i,j}$ is only related to $b_{k,j}, k = 1, 2, \dots, d_r, k \neq i$. Under this scenario, applying the MAP decoding on this variable node is therefore equivalent to applying the MAP decoding on z single parity-check (SPC) code defined by the z rows of $\mathbf{S}$, which is explained as follows.

Assume that we are to calculate the message for c_1. Consider the probability domain message is under concern for convenience. Denote $\boldsymbol{\tau}_i = (\tau_i(\gamma_0), \tau_i(\gamma_1), \dots, \tau_i(\gamma_{q-1}))$ as the message from the $i^{\,th}$ variable node, which is calculated based on the bits $(b_{i,1}, b_{i,2}, \dots, b_{i,z})$. To calculate the message for c_1, we compute

$$\tau_1(\gamma_j) = \sum_{\substack{(c_2, c_3, \dots, c_{d_r}) : \\ \sum_{k=2}^{d_r} c_k = \gamma_j}} \prod_{k=2}^{d_r} \tau_k(c_k). \tag{5.24}$$

Furthermore, denote $\theta_{i,j}(a), a \in GF(2)$ as the message for the bit $b_{i,j}$; and a function $M_i(s) \in GF(2), s \in GF(q), i = 1, 2, \dots, p$ is defined as : $M_i(s)$ represents the value of the $i^{\,th}$ bit in the binary representation of the finite field element s. Apparently,

$$\tau_i(\gamma_j) \;\; = \;\; \prod_{k=1}^{z} \theta_{i,k}(M_k(\gamma_j)),$$

$$\theta_{i,k}(a) \;=\; \sum_{\gamma_j : M_k(\gamma_j)=a} \tau_i(\gamma_j). \tag{5.25}$$

Note that the above equations are valid because each $b_{i,j}$ affects only the $j^{\,th}$ row in $\mathbf{S}$, and hence the calculated message for c_k can be treated separately as z independent messages for $b_{k,j}, j = 1, 2, \ldots, z$; i.e., $b_{k,j}$ is only affected by the $j^{\,th}$ row in $\mathbf{S}$, each of the rows contains a number of d_r involved bits $(b_{1,j}, b_{2,j}, \ldots, b_{d_r,j})$, but any two rows contains no common effective bit positions. *For codes with weights other than* 1*, before the* 1 st *iteration, the messages for bits within a symbol are also independent; however, after the first round of messages updating process, the bit messages become dependent. Hence,* (5.25) *is not valid for codes with non-identical weights.*

Therefore,

$$\begin{aligned}
\theta_{1,l}(a) \;&=\; \sum_{\gamma_j : M_l(\gamma_j)=a} \tau_1(\gamma_j) \\
&=\; \sum_{\gamma_j : M_l(\gamma_j)=a} \left(\sum_{\substack{(c_2, c_3, \ldots, c_{d_r}) : \\ \sum_{k=2}^{d_r} c_k = \gamma_j}} \prod_{k=2}^{d_r} \tau_k(c_k) \right) \\
&=\; \sum_{\gamma_j : M_l(\gamma_j)=a} \left[\sum_{\substack{(c_2, c_3, \ldots, c_{d_r}) : \\ \sum_{k=2}^{d_r} c_k = \gamma_j}} \prod_{k=2}^{d_r} \left(\prod_{h=1}^{z} \theta_{k,h}(M_h(c_k)) \right) \right] \\
&=\; \sum_{\gamma_j : M_l(\gamma_j)=a} \sum_{\substack{(\mathbf{b}_2, \mathbf{b}_3 \ldots, \mathbf{b}_{d_r}) : \\ \sum_{k=2}^{d_r} b_{k,1} = M_1(\gamma_j) \\ \sum_{k=2}^{d_r} b_{k,2} = M_2(\gamma_j) \\ \vdots \\ \sum_{k=2}^{d_r} b_{k,z} = M_z(\gamma_j)}} \theta(b_{2,1})\theta(b_{2,2}) \ldots \theta(b_{d_r,z-1})\theta(b_{d_r,z}).
\end{aligned} \tag{5.26}$$

(5.26) implies that $\theta_{1,l}(a)$ is calculated with the following two steps :

1. For each fixed γ_j, the inner summation enumerates all the combination of bits that sums up to the binary representation of γ_j when multiplied by the parity-check matrix $\mathbf{S}$;
2. the outer summation enumerates all those γ_j whose $l^{\,th}$ bit is a.

As mentioned earlier, only $(b_{2,l}, b_{3,l}, \ldots, b_{d_r,l})$ affects the calculation for $\theta_{1,l}(a)$ whilst others can be taken arbitrarily, (5.26) is therefore simplified as

$$\theta_{1,l}(a) = \sum_{\substack{(\mathbf{b}_1, \mathbf{b}_2 \ldots, \mathbf{b}_{d_r}) : \\ \sum_{k=2}^{d_r} b_{k,l} = a}} \theta(b_{2,1})\theta(b_{2,2}) \ldots \theta(b_{d_r,z-1})\theta(b_{d_r,z}), \tag{5.27}$$

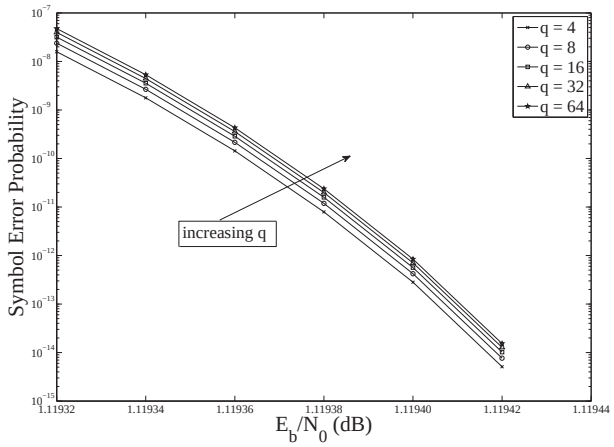
which is precisely the MAP decoding equation for the bit $b_{1,l}$ in a single parity-check code composed of bits $(b_{1,l}, b_{2,l}, \ldots, b_{d_r,l})$. Therefore, we may conclude that decoding the q-ary code with identical weights over $GF(q)$ is equivalently to decode z identical binary codes. Hence, q-ary codes with identical weights has the same thresholds as binary codes with the same graph structure.

Using the above property for the codes with identical weights, in calculating the decoding thresholds for this specific subset of codes, instead of tracking the p.d.f. of a message with message alphabet of size 2^{gq}, we only need to track the a message with alphabet size 2^g for each of the z bits. Furthermore, it is obvious that the messages corresponding to different bits in a single symbol have the same probability density function under the BP decoding algorithm for q-ary codes with identical weights.

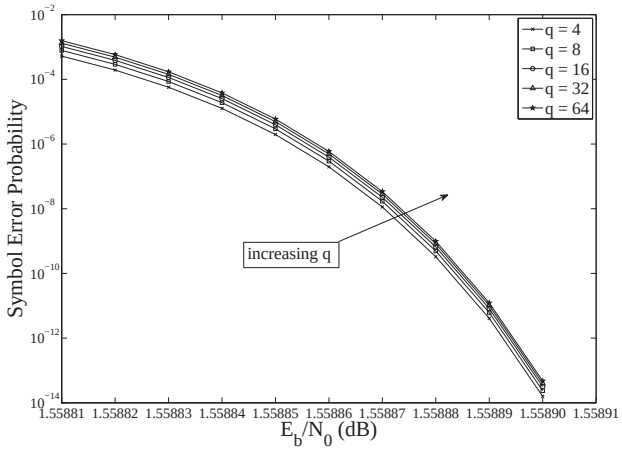
It is worth noting that q-ary codes with identical weight values have higher symbol error rates (SERs) than the binary codes simply due to that each of the z bits in a symbol potentially cause a symbol error.

We calculate the symbol error probability based on the above mentioned simplification for q-ary codes with identical weights using quantized density evolution for binary codes as described in (Chung *et al.*, 2001a). The results are given in Fig. 5.2. It is observed the symbol error probabilities monotonically increases with q for fixed code rate (whilst the bit error probability remains the same). Fig. 5.2 also demonstrates that codes with denser graphs (larger degrees) tend to have a sharper threshold region, e.g., it takes the rate 3/6 code ensemble 0.0001 dB to drop from 10^{-7} to 10^{-14} whilst it takes the rate 5/10 code ensemble less than 0.00002 dB.

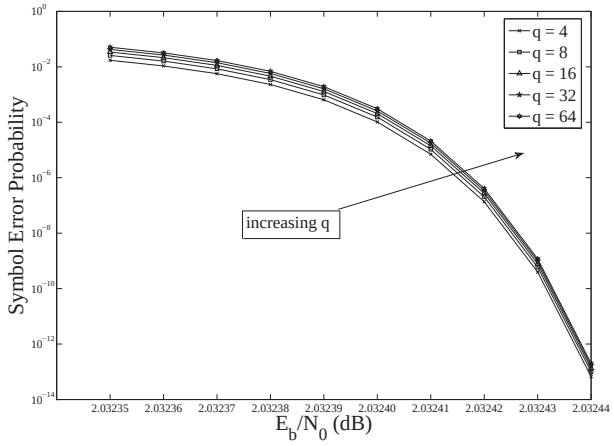
Note that the results in Fig. 5.2 is for the identical weight codes, i.e., when bits within a symbol do not affect each other. However, when the weights are randomly selected, i.e., when the $\mathbf{S}$ is denser, the bits within a single symbol become dependent. Therefore, we may expect the error of one bit to affect the other bits in the same symbol ; hence, the above simplification cannot be applied to calculate the error probability of codes with randomly selected weights. That is to say, the symbol error probability curves in Fig. 5.2 *cannot* be used to infer the influence of increasing the value of q on the error performances of our codes when their weights are randomly selected ; in order to calculate the decoding thresholds for codes with random weights, we still need to apply density evolution with the Gaussian approximation (Li *et al.*, 2009) as in Chapter 3.



(a) Rate 3/6



(b) Rate 4/8



(c) Rate 5/10

Figure 5.2 Symbol error probability as a function of E_b/N_0 and q for rate $3/6, 4/8, 5/10$ codes. iteration number =150

Discussion: Effects of weight selection

The analysis of the influence of weight selection on the error performances of the q-ary codes can be summarized as follows.

- Section 5.2.1 states that the identical selection of weights is harmful when there are erroneous messages for it propagates the error in one variable node to others *in the same cycle* while the randomly selected weights help to dilute the erroneous messages.
- Section 5.2.2 discusses how weight selection affect the message updating process at the constraint node, it is shown that the random selection may improve the error performance for it scatters the effect of the erroneous message from a variable node onto different symbols of the variable nodes *connected to the same constraint node* (in the same parity-check equation).
- In Section 5.2.3, it is stated that the randomly selected weights have the negative effect that bits within a symbol become dependent and therefore an erroneous bit message may harm other bits *within the same symbol*, while an erroneous bit message has no effects on other bits in the same symbol when using identical weights. However, it *must be noted* that when all the weights are equal to 1, the equivalent binary parity-check matrix $\mathbf{S}$ of a single constraint node represents a weak parity-check code for it is composed of d_c diagonal matrices. On the other hand, the merits of the q-ary alphabets come from a denser $\mathbf{S}$, when the weights are randomly selected, a bit is affected by other bits in the same symbol; although a bit is potentially influenced by erroneous bit messages in the same symbol, a denser $\mathbf{S}$ represents a strong binary parity-check code (a single constraint node), improving the error performances of the q-ary codes.

According to the discussion, the selection of identical weights has influences over various perspectives of the iterative decoding process. From both the simulation results and the analysis, we conclude that it is beneficial to apply randomly selected weights rather than identical weight for our q-ary CDO codes.

5.3 Summary

In this chapter, we have presented discussion on the weight selection of the q-ary CDO codes. As observed in our simulations, the codes with their weights randomly selected enjoy error performances superior to the ones with identical weights. Our analysis shows that the random selection of weights benefit the decoding through message processing in the Tanner

graphs, which is explored in detail through the message updating in the constraint node. Furthermore, in analyzing the equivalent binary parity-check matrices of the constraint nodes, it is shown that although identically selected weights prevent the erroneous bit messages from affecting other bits in the same symbol, the random selection of weights improves the error performances of the q-ary codes by increasing the density of the equivalent binary parity-check matrices of the constraint nodes. Hence, we conclude that while the optimal scheme is to select the weights according to the specific cycle features of the Tanner graphs, the random selection is a acceptable alternative.

CHAPTER 6

Conclusion and suggestion for future work

6.1 Conclusion

The main focus of the thesis is the extension of the binary convolutional doubly orthogonal codes with the finite fields alphabets, referred to as the q-ary non-recursive and recursive CDO codes, and its effects. Various aspects such as the error performances, the decoding thresholds, and the impacts of the quantized messages in the belief propagation algorithm have been investigated. It is shown in our simulation that the extension serves to improve the error performances of this set of codes. The approximate bounds on the free distances of the binary convolutional codes has also been generalized to embrace the q-ary CDO codes, which partially explains the error performance differences among the codes with different alphabet sizes.

The initial investigation is on the generalization of the non-recursive q-ary CDO codes with single-shift-register structures using finite field alphabets. The new set of codes can be decoded with either the generalized iterative threshold decoding algorithm or the belief propagation algorithm. Simulation results demonstrate that shifting to finite fields of higher order has the effect of lowering the bit error rates of these codes in relatively high E_{b}/N_0 region. The most pronounced benefits is at the error floors of the non-recursive CDO codes. Depending on the selection of codes and the scaling parameter in the decoding algorithms, codes defined over finite fields of higher order achieve error floors 1 or 2 orders of magnitudes lower than those of the binary codes. The extension of the CDO codes onto the finite fields potentially reduces the decoding latency for they are capable of achieving satisfactory error performances with much smaller memory order. The error performances of the simplified non-recursive q-ary CDO codes are also investigated. By loosening the doubly orthogonal conditions, these codes achieve much smaller decoding latency owing to their smaller memory order ; only slight performance degradation has been observed with the simplified codes as compared with the non-simplified ones, making them more attractive than the latter in real applications. The decoding complexity, however, increases with the field order, which is the price for the improvements in the error performances. Furthermore, due to its unique message processing at the variable nodes, the iterative threshold decoding somehow underperforms the BP algorithm for this set of codes ; however, it also greatly accelerates the convergence of the decoding process, requiring much smaller number of iterations to achieve reasonable error

performances, hence reducing the decoding latency of these codes. Specifically, the iterative threshold decoding algorithm requires 8 iterations to achieve satisfactory error performances for our codes under concern whilst the BP decoding algorithm requires 50. The decoding thresholds of this set of codes under the BP decoding algorithm are evaluated using the density evolution with Gaussian approximation. Numerical calculations showed that the decoding thresholds of the non-recursive q-ary CDO codes are affected by such factors as field order, number of connections from the shift register, and the scaling parameter in the decoding process.

For the q-ary codes under discussion, we have applied randomly selected weights along the edges in their Tanner graphs. Simulation results showed that the codes with random weights substantially outperform those with identical weights. Through our analysis of the decoding process, it is demonstrated that the identical selection of weights has negative impacts on various aspects of the decoding procedure; e.g., the message passing within short-length cycles, the constraint node updating, and the relation between bits within the same symbol. We conclude that while it may be preferable to construct codes with their weights optimized according to their particular Tanner graphs, the random selection provides a simple feasible alternative, providing satisfactory error performances for this set of codes.

Following the investigation into the q-ary non-recursive CDO codes, the finite field alphabets are also applied on the recursive CDO codes based on protographs. The BP algorithm is applied for this set of codes. Simulation results demonstrate that the benefit of moving to higher order fields is more prominent when E_b/N_0 is relatively higher. While codes with lower-order fields perform slightly better in lower E_b/N_0 region, they are swiftly surpassed by those with higher-order fields as E_b/N_0 increases. This is because in lower E_b/N_0 region, a large value for q prevents the 'correct' message from dominating the decoding process due to the large number of coordinates in a single message; when E_b/N_0 is relatively high, the error performances for codes with large q improves quickly due to the denser structures in their equivalent binary parity-check matrix for a single constraint node. Similar to the cases with the non-recursive CDO codes, recursive CDO codes with small protographs defined over higher-order fields may achieve comparable error performances as the binary ones with much larger protographs. The benefit is the decrease in the decoding latency for the memory order of the recursive CDO codes increases rapidly with the size of the protograph under the doubly orthogonal conditions. Furthermore, the performances of this set of codes under the quantized BP algorithm are also examined with computer simulation. Although the FFT and Log-FFT-based decoding algorithms greatly reduce the computation complexity, they do not lend themselves to the quantized message alphabets due to numerical reasons; i.e., the FFT algorithm requires very fine quantization for those messages close to 0 while the

Log-FFT algorithm requires that for messages with large magnitudes. Hence, the Log-domain decoding with quantized message alphabets is implemented instead for the decoding of this set of codes. It is observed that 9-bit quantization scheme is quite sufficient for these codes to achieve comparable error performances as the floating point decoding scheme under 50 iterations. However, when more iterations are performed (100), the gap between the quantized decoder and the floating point decoder is somehow enlarged. After a larger number of iterations, e.g., when the number of iterations is above 50, the messages tend to converge to a smaller range of values, therefore finer quantization is required to distinguish between these messages.

In summary, by shifting from the binary code alphabet to finite field alphabets, the error performances of the non-recursive and recursive CDO codes is improved, the effects of which is more pronounced in relatively higher E_{b}/N_0 region. The improvements make it possible to use q-ary CDO codes with smaller constraint lengths in place of binary ones with longer constraint lengths, the benefit of which is the reduced decoding complexity. However, due to the increased decoding complexity of the q-ary CDO codes, it is more appropriate to apply these codes when complexity is less of a concern.

The distance properties of the q-ary linear codes are examined through the upper and lower bounds on the minimum distances of the q-ary block codes and the free distances for the q-ary convolutional codes. For the equiprobable code ensembles of the q-ary block codes, we derived an upper bound on the minimum distance. The main focus of this part of work is on the q-ary convolutional codes. The lower bound on the free distances for the q-ary convolutional codes considers the existence of codes within the entire code ensemble which enjoys a free distance at least as large as the bound, while the upper bound on the free distances applies to any q-ary convolutional code. In our derivation, we applied the bound on the probabilities of the sum of random variables, which had been derived using the Chernoff bound (Wozencraft and Reiffen, 1961), in upper bounding the number of codewords in our discussion on the lower bounds on the free distances for q-ary convolutional codes. The numerical evaluation of the bounds on the free distances for our q-ary recursive CDO codes demonstrates that the lower bound on the free distances of a q-ary recursive CDO codes increases with q while other parameters are fixed, although the improvement diminishes with increasing q. The lower bounds are also applied in the comparison of the error probabilities of the q-ary recursive CDO codes, partially accounting for the performance differences among codes with different q values.

The q-ary CDO codes may find their applications when low bit error rate is required, especially under harsh channel conditions, such as optical fibre communication, space and satellite communications, etc. ; also they may be suitable candidates when the information

source fits naturally into non-binary alphabets, e.g., letters and audio signals; furthermore, the q-ary CDO codes may also be implemented with multiple-level modulations, e.g. PSK and QAM modulations.

6.2 Suggestion for future work

6.2.1 Simplified q-ary CDO codes

In our discussion on the non-recursive q-ary CDO codes, it is shown that loosening the doubly orthogonal conditions produces a group of codes capable of achieving much smaller decoding latency with minor compromise in the error performances. Similar effects could be expected with recursive codes when their corresponding doubly orthogonal conditions are relaxed. In actual applications, the simplified codes are of higher value than non-simplified ones due to their smaller latencies in decoding. Various aspects investigated for the non-simplified codes in this thesis, e.g., the decoding thresholds, the effects of message quantization, etc. could also be applied in the study of the simplified codes. However, owing to the larger number of short-length cycles in the Tanner graphs of the simplified codes, it may not be appropriate to apply the density evolution algorithm in analyzing the decoding thresholds of these codes.

6.2.2 Generalized CDO codes

The codes defined over the finite fields could be regarded as a specific case of the generalized binary codes. It has been demonstrated that a constraint node of a q-ary code is actually equivalent to a binary block code (referred to as a subcode) of length zd_r. However, the choice of the subcodes in a graph-based code is actually arbitrary. As in Section 2.2.2, the subcodes for both variable nodes and constraint nodes may be selected in accordance with the desired rate and graph structure. Therefore, it is interesting to consider binary CDO codes with their constraint nodes defined by block codes other than single parity checks, providing flexibility in the design of the codes.

The encoding of the generalized codes may also be implemented in a systematic manner. Considering the protograph-based design, in (2.3) the value of $h_{i,j}$ indicates whether a connection exists between the $i^{\,th}$ variable node and $j^{\,th}$ constraint node. We take the first K variable nodes as inputs and the rest $N-K$ as outputs, and suppose the subcode at the constraint nodes are all identical block codes with rate k/n; in each of the constraint nodes, $m=n-k$ bits would be taken as parity bits for the subcode. In our design, we may specify that the $(K+(j-1)m+1)^{\,th}$ through the $(K+jm)^{\,th}$ variable nodes are connected to the $j^{\,th}$ constraint node, regarded as the parity bits of that subcode. As an example, The

position of the variable nodes taken as parity checks for $m = 2$ and $N - K = 6$ are illustrated as follows.

$$\mathbf{H}^T(D) = \left[\begin{array}{ccc} \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \vdots & K \text{ rows} & \vdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \hline & & \\ \star & \cdots & \cdots \\ \star & \cdots & \cdots \\ \cdots & \star & \cdots \\ \cdots & \star & \cdots \\ \cdots & \cdots & \star \\ \cdots & \cdots & \star \end{array}\right]. \tag{6.1}$$

$\star$ positions represent the variable nodes corresponding to the parity bits. They are placed as in (6.1) in order to facilitate the systematic encoding. For the convenience of encoding, it is preferable that the $\star$ positions have the smallest delay among all the connected variable nodes in the same constraint node ; so that it is unnecessary to wait for later information bits when calculating the parity bits. Just as in the parity-check matrix for a normal CDO code, none of the rows in (6.1) contains more than one $\star$ for the value of a parity variable node is determined by a single subcode (constraint node) in systematic encoding.

For the codes with their constraint nodes generalized in the above stated manner, the constraint nodes process the incoming messages and provide for each connected variable node the latest version of the reliabilities for every element in the code alphabet. There are essentially two approaches of accomplish the task.

- For a constraint node (a subcode), MAP decoding is applied to calculate the messages for each subcode bits. It is always simpler to apply the BCJR algorithm based on the trellis of the subcode.
- Since the subcode is a linear block code itself, the decoding of the subcode may be carried out by the BP decoding based on the subcode (local) Tanner graph.

The second approach is advantageous over the first one in terms of complexity. However, one obvious defect exists : usually the subcodes are not sparse codes and therefore contain many short-length cycles in their Tanner graphs. Hence, it is unnecessary to perform too many iterations (local iterations) within a subcode before it transmits its messages back to the incident variable nodes. Promising candidate of the subcodes may include Hamming codes,

and multi-dimensional SPC product codes (Rankin and Gulliver, 2001), etc.

The thresholds of the generalized codes may also be evaluated with density evolution. Furthermore, we are again interested in quantized decoders for real implementation. For the situation where BCJR algorithm is applied with the constraint nodes, the calculation of the thresholds is relatively direct. However, with a local BP algorithm based on the Tanner graphs of the subcodes, further exploration is required in order to evaluate the probability density functions of the messages.

6.2.3 Group codes with appropriate modulation schemes

It is known that for codes defined with finite fields modulated with multi-level signals, the channel is no longer considered symmetrical, i.e., the choice of information symbols affects the error performance of the codes. In order to avoid the problem, it is desirable to consider modulation schemes matched to specific code alphabets. It is known that for ring codes, MPSK modulation is a suitable candidate (Forney, 1991). There are some general criteria for determining whether a set of signals is matched to a specific algebra (Ingemarsson, 1973) ; however, for a given code, it is not always possible to find the signal sets satisfying the requirements.

In the study of the CDO codes, it is of interest to examine the CDO codes defined with various algebras with an aim to find the suitable (in terms of equal error protection) modulation schemes.

REFERENCES

BAHL, L., COCKE, J., JELINEK, F. and RAVIV, J. (1974). Optimal decoding of linear codes for minimizing symbol error rate (Corresp.). *IEEE Transactions on Information Theory*, 20, 284–287.

BASSALYGO, L. A. (1965). New Upper Bounds for Error Correcting Codes. *Problemy Peredachi Informatsii (Problems of Information Transmission)*, 1, 31–35.

BERROU, C., GLAVIEUX, A. and THITIMAJSHIMA, P. (1993). Near Shannon limit error-correcting coding and decoding : Turbo-codes. 1. *IEEE International Conference on Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record,*. vol. 2, 1064 –1070 vol.2.

BOSE, R. C. and RAY-CHAUDHURI, D. K. (1960). On A Class of Error Correcting Binary Group Codes. *Information and Control*, 3, 68–79.

BOUTROS, J., POTHIER, O. and ZEMOR, G. (1999). Generalized low density (Tanner) codes. *IEEE International Conference on Communications, 1999. ICC '99.* vol. 1, 441 –445 vol.1.

CARDINAL, C., HACCOUN, D. and GAGNON, F. (2003). Iterative threshold decoding without interleaving for convolutional self-doubly orthogonal codes. *IEEE Transactions on Communications*, 51, 1274 – 1282.

CARDINAL, C., HE, Y.-C. and HACCOUN, D. (2008). A New Approach for the Construction of Powerful LDPC Convolutional Codes. *IEEE Vehicular Technology Conference, 2008. VTC Spring 2008.* 1176 –1180.

CARDINAL, C., ROY, E. and HACCOUN, D. (2009). Simplified convolutional self-doubly orthogonal codes : search algorithms and codes determination. *IEEE Transactions on Communications*, 57, 1674 –1682.

C.DAVEY, M. (1999). *Error-Correction Using Low-Density Parity-Check Codes*. Cambridge University.

CHUNG, S.-Y., FORNEY, G.D., J., RICHARDSON, T. and URBANKE, R. (2001a). On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications Letters*, 5, 58–60.

CHUNG, S.-Y., RICHARDSON, T. and URBANKE, R. (2001b). Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation. *IEEE Transactions on Information Theory*, 47, 657 –670.

COSTELLO, D. (1974). Free distance bounds for convolutional codes. *IEEE Transactions on Information Theory*, 20, 356–365.

DAVEY, M. and MARKAY, D. (1998). Low density parity check codes over GF(q). *IEEE Communication Letters*, 2, 165–167.

DECLERCQ, D. and FOSSORIER, M. (2007). Decoding algorithm for nonbinary LDPC codes over GF(q). *IEEE Transactions on Communications*, 55, 633–643.

DEGROOT, M. H. (2004). *Optimal Statistical Decisions*. John Wiley & Sons, Inc.

DJURDJEVIC, I., XU, J., GHAFFAR, K. and LIN, S. (2003). A Class of Low-Density Parity-Check Codes Constructed Based on Reed-Solomon Codes with Two Information Symbols. *Book Chapter of "Lecture Notes in Computer Science"*. Springer Press.

FELTSTROM, A., TRUHACHEV, D., LENTMAIER, M. and ZIGANGIROV, K. (2009). Braided block codes. *IEEE Transactions on Information Theory*, 55, 2640–2658.

FORNEY, G.D., J. (1991). Geometrically uniform codes. *IEEE Transactions on Information Theory*, 37, 1241 –1260.

FOSSORIER, M. (2004). Quasi-Cyclic Low-Density Parity-Check Codes From Circulant Permutation Matrices. *IEEE Transactions on Information Theory*, 50, 1788–1793.

FOSSORIER, M., MIHALJEVIC, M. and IMAI, H. (1999). Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on Communications*, 47, 673–680.

GALLAGER, R. (1963). *Low-Density Parity-Check Codes*. Cambridge University.

HACCOUN, D., CARDINAL, C. and GAGNON, F. (2005). Search and determination of convolutional self-doubly orthogonal codes for iterative threshold decoding. *IEEE Transactions on Communications*, 53, 802 – 809.

HE, Y.-C. and HACCOUN, D. (2005). An analysis of the orthogonality structures of convolutional codes for iterative decoding. *IEEE Transactions on Information Theory*, 51, 3247 – 3261.

HE, Y.-C., HACCOUN, D. and CARDINAL, C. (2009). Comparison of low complexity fast iterative decoding techniques for convolutional self-doubly-orthogonal codes. *Proceedings of IEEE 69th Vehicular Technology Conference, 2009*. 1–5.

HOCQUENGHEM, A. (1959). Codes correcteurs d'erreurs. *Chiffres*, 2, 147–156.

HU, X., ELEFTHERIOU, E. and ARNOLD, D. (2001). Progressive Edge-Growth Tanner Graphs. *Proceedings of IEEE GLOBECOM 2001*. 995–1001.

INGEMARSSON, I. (1973). Commutative group codes for the Gaussian channel. *IEEE Transactions on Information Theory*, 19, 215 – 219.

JIMENEZ FELSTROM, A. and ZIGANGIROV, K. (1999). Time-varying periodic convolutional codes with low-density parity-check matrix. *IEEE Transactions on Information Theory*, 45, 2181 –2191.

JOHNSON, S. (1962). A new upper bound for error-correcting codes. *IRE Transactions on Information Theory*, 8, 203–207.

KOU, Y., LIN, S. and FOSSORIER, M. (2001). Low-Density Parity-Check Codes Based on Finite Geometries : A Rediscovery and New Results. *IEEE Transactions on Information Theory*, 47, 2711–2736.

KOWARZYK, G., BELANGER, N., HACCOUN, D. and SAVARIA, Y. (2012). Efficient search algorithm for determining optimal r=1/2 systematic convolutional self-doubly orthogonal codes. *IEEE Transactions on Communications*, 60, 3–8.

KOWARZYK, G., BELANGER, N., HACCOUN, D. and SAVARIA, Y. (2013). Efficient parallel search algorithm for determining optimal r=1/2 systematic convolutional self-doubly orthogonal codes. *IEEE Transactions on Communications*, 61, 865–876.

KOWARZYK, G., SAVARIA, Y. and HACCOUN, D. (2008). Searching for short-span convolutional doubly self-orthogonal codes : A parallel implicitly-exhaustive -search algorithm. *2008 Canadian Conference on Electrical and Computer Engineering*. 001659–001662.

KSCHISCHANG, F., FREY, B. and LOELIGER, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47, 498 –519.

KURKOSKI, B., YAMAGUCHI, K. and KOBAYASHI, K. (2007). Density evolution for gf(q) ldpc codes via simplified message-passing sets. *Information Theory and Applications Workshop, 2007*. 237–244.

LENTMAIER, M. and ZIGANGIROV, K. (1999). On generalized low-density parity-check codes based on Hamming component codes. *IEEE Communications Letters*, 3, 248 –250.

LI, G., FAIR, I. and KRZYMIEN, W. (2009). Density Evolution for Nonbinary LDPC Codes Under Gaussian Approximation. *IEEE Transactions on Information Theory*, 55, 997 –1015.

LIDL, R. and NIEDERREITER, H. (1997). *Finite Field*. Cambridge University Press.

LIN, S. and COSTELLO, D. J. (2004). *Error Control Coding (2nd Edition)*. Pearson Prentice Hall.

LOELIGER, H.-A. (1991). Signal sets matched to groups. *IEEE Transactions on Information Theory*, 37, 1675 –1682.

MACKAY, D. (1999). Good Error Correcting Codes Based on Very Sparse Matrices. *IEEE Transactions on Information Theory*, 45, 399–431.

MACKAY, D. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge Unversity Press.

MASSEY, J. L. (1963). *Threshold Decoding*. Massachusetts Inst. of Tech., Cambridge Research Lab of Electronics.

MILADINOVIC, N. and FOSSORIER, M. (2005). Generalized LDPC codes with Reed-Solomon and BCH codes as component codes for binary channels. *IEEE GLOBECOM 2005*. vol. 3, 6 pp.

MITCHELL, D., PUSANE, A., ZIGANGIROV, K. and COSTELLO, D. (2008). Asymptotically good LDPC convolutional codes based on protographs. *IEEE International Symposium on Information Theory, 2008. ISIT 2008*. 1030 –1034.

PAULUZZI, D. and BEAULIEU, N. (2000). A Comparison of SNR Estimation Techniques for the AWGN Channel. *IEEE Transactions on Information Theory*, 48, 1681–1691.

PLOTKIN, M. (1960). Binary codes with specified minimum distance. *IRE Transactions on Information Theory*, 6, 445–450.

RANKIN, D. and GULLIVER, T. (2001). Single parity check product codes. *IEEE Transactions on Communications*, 49, 1354 –1362.

RATHI, V. and URBANKE, R. (2002). Density Evolution, Thresholds and Stability Condition for Non-binary LDPC Codes. *EPFL, CH-1015, Lausanne*.

REED, I. S. and SOLOMON, G. (1960). Polynomial Codes over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8, 300–304.

RICHARDSON, T. and URBANKE, R. (2001). The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding. *IEEE Transactions on Information Theory*, 47, 599–618.

RLITSKY, A., URBANKET, R., VISWANATHAN, K. and ZHANG, J. (2002). Stopping Sets and the Girth of Tanner Graphs. *Proceedings of International Symposium on Information Theory, Lausanne, Switzerland, 2002*.

ROY, E. (2006). *Recherche et Analyse de Codes Convolutionnels Doublement Orthogonaux Simplifiés au Sens Large, M.E. Thesis*. Dept. Electrical Engineering, École Polytechnique de Montréal.

ROY, E. (2011). *Étude des Propriétés des Codes Convolutionnels Récursifs Doublement-Orthogonaux, Ph.D. Thesis*. Dept. Electrical Engineering, École Polytechnique de Montréal.

ROY, E., CARDINAL, C. and HACCOUN, D. (2010). Recursive convolutional codes for time-invariant LDPC convolutional codes. *Proceedings of IEEE International Symposium on Information Theory (ISIT) 2010*. 834 –838.

RYAN, W. (2004). An Introduction to LDPC Codes. *CRC Handbook for Coding and Signal Processing for Recoding Systems*. CRC Press.

SAVIN, V. (2008). Min-Max decoding for non binary LDPC codes. *Proceedings of IEEE International Symposium on Information Theory (ISIT) 2008*. 960 –964.

SHEN, X. H., HACCOUN, D. and CARDINAL, C. (2013). Convolutional doubly orthogonal codes over GF(q). *IET Communications*, 7, 1126–1132.

SINGLETON, R. (1963). Maximum Distance Q-Nary Codes. *IEEE Transactions on Information Theory*, 10, 116–118.

SLEPIAN, D. (1968). Group codes for the Gaussian channel (Abstr.). *IEEE Transactions on Information Theory*, 14, 242.

SONG, H. and CRUZ, J. (2003). Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording. *IEEE Transactions on Magnetics*, 39, 1081 – 1087.

TANNER, R. (1981). A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27, 533–547.

TANNER, R., SRIDHARA, D., SRIDHARAN, A., FUJA, T. and COSTELLO, D.J., J. (2004). LDPC block and convolutional codes based on circulant matrices. *IEEE Transactions on Information Theory*, 50, 2966 – 2984.

TANNER, R. M., SRIDHARA, D. and FUJA, T. (2001). A Class of Group-Structured LDPC Codes. *Proceedings of ICSTA, Ambleside, U.K.*

TAVARES, M. B., ZIGANGIROV, K. S. and FETTWEIS, G. P. (2007). Tail-Biting LDPC Convolutional Codes. *IEEE International Symposium on Information Theory, 2007. ISIT 2007.* 2341 –2345.

TEN BRINK, S. (1999). Convergence of iterative decoding. *Electronics Letters*, 35, 806–808.

VITERBI, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13, 260–269.

WANG, Y. and FOSSORIER, M. (2006). Doubly Generalized LDPC Codes. *Proceedings of International Symposium on Information Theory, Seatle, USA*. 669–673.

WOZENCRAFT, J. M. and JACOBS, I. M. (1965). *Principles of Communication Engineering*. John Wiley & Sons, Inc.

WOZENCRAFT, J. M. and REIFFEN, B. (1961). *Sequntial Decoding*. Published jointly by the Technology Press of the Massachussets Institute of Technology and Wiley.

WYMEERSCH, H., STEENDAM, H. and MOENECLAEY, M. (June 2004). Log-Domain Decoding of LDPC Codes over GF(q). *Proceedings of IEEE International Conference on Communications*. Paris, France, 772–776.

APPENDIX A

Bounds on The Distances for q-ary Codes

In this appendix, we analyze the distance properties of q-ary convolutional codes. Particularly, the bounds on the minimum distance of q-ary block codes and the free distance of q-ary convolutional codes are derived based on those on the binary codes. The lower bound on the free distances of q-ary convolutional codes provides another explanation on the improvements in the error performances of our codes as larger alphabet sizes are used.

The minimum distances for block codes and the free distances for convolutional codes are among the most essential measures for evaluating the error performances of the group of linear codes decoded with maximum likelihood (ML) algorithm. Whilst the exact calculation of the minimum distances for linear codes with very long constraint lengths are still inapproachable with existing methods, the upper and lower bounds on the minimum distance prove to be acceptable alternatives in assessing these codes. However, while these bounds had been rigorously examined for linear codes defined over the binary field $GF(2)$, those for the codes defined over $GF(q)$, with $q > 2$, attracted far less attention. This appendix extends various bounds on the minimum distances on the binary linear codes obtained by Gallager and by Costello onto the general finite fields $GF(q)$ for $q > 2$.

In this appendix, an upper bound on the minimum distances for the q-ary block codes is derived using Gallager's approach (Gallager, 1963) for the equiprobable code ensembles. The main concern of our work, however, is on the convolutional codes. In the decoding process over noisy channels, the distance between two encoded sequences is the main parameter for determining the pairwise error probability, which is the probability that any one of them is mistakenly recognized to be the other. Particularly, the minimum distance $d_{\min}$ is important for convolutional codes with *fixed-length* code sequences decoded with threshold decoding; whilst the free distance d_{free} (Lin and Costello, 2004) concerns the probabilistic decoding when we consider encoded sequences with *undetermined lengths*. In the study of q-ary convolutional codes, we are more interested in d_{free} for a single error symbol affects a range of other symbols of undetermined lengths in the decoding process. Costello (Costello, 1974) had derived upper and lower bounds for the free distances of convolutional codes over $GF(2)$, by examining the existence of codes within the entire code ensemble that satisfies the bounds. We followed his rationale in our derivation of the bounds for the q-ary convolutional codes. Although our concentration is on the free distance d_{free}, the bounds on the minimum distances for the q-ary convolutional codes have also been derived following Massey's work on

the binary codes (Massey, 1963), serving as a first step towards the lower bounds on d_{free}. Note that in our derivation, we applied the bound on the probabilities of the sum of a series of random variables, which was derived using the Chernoff bound (Wozencraft and Reiffen, 1961; Wozencraft and Jacobs, 1965).

It is demonstrated in previous chapters of the thesis that the extension of the CDO codes onto $GF(q)$ for $q > 2$ serves to further improve the codes' performances. In latter part of this appendix, we also examine the asymptotic gain for codes with larger alphabets using the derived lower bounds on d_{free}. Although the gain actually observed does not coincide with the computed values due to various approximations made in the derivation of the asymptotic gain, the benefits on the free distances brought by the extension of code alphabets also shed some light on the underlying causes of the improvements in the error performances of q-ary codes.

The minimum distances $d_{\min}$ for q-ary codes has a similar definition as for the binary codes. It applies both to code sequences of block codes and to fix-length segments of code sequences generated by convolutional codes.

Definition 2 *The distance between two codewords is the number of symbols in which they differ; the minimum distance d_{min} of a code is the smallest distance between any two of its codewords; where the weight of a sequence is the number of its non-zero symbols.*

Note that for a linear code, $d_{\min}$ is equal to the minimum weight of all its codewords.

A.1 On the minimum distances of block codes

In this section, we extend the bound on the $d_{\min}$ of binary block codes (Gallager, 1963) over finite fields $GF(q)$. Particularly, we present an upper bound on the probability for the ratio of $d_{\min}$ to the code length to be smaller than some fixed value using the equiprobable ensembles of q-ary block codes.

Denote n as the code length of a parity-check code, and R as its coding rate. The *equiprobable ensemble* is defined as the set of codes corresponding to all the $[n(1-R)]\times n$ parity-check matrices filled independently with the elements in $GF(q)$ with equal probabilities. The *weight* of a sequence of symbols is the number of its non-zero elements.

Similar to the methods in (Gallager, 1963), over the equiprobable ensemble for the block codes, we wish to examine the probability that $d_{\min}$ is smaller than a certain value δn, i.e., $\text{Prob}(d_{\min} \leq \delta n)$ for $0 < \delta < 1$.

The number of sequences over $GF(q)$ of length n and weight l is given by $\binom{n}{l}(q-1)^l$; therefore, the number of non-zero sequences of weight less than or equal to a certain number

Δ is given by

$$\sum_{l=1}^{\Delta} \binom{n}{l} (q-1)^l. \tag{A.1}$$

For the binary case in (Gallager, 1963), i.e., when $q = 2$, (A.1) was bounded by the sum of a geometric series. However, for code alphabets over $GF(q)$, we need a different approach.

Lemma 1 *For positive integers n, q and $\Delta < n$,* (A.1) *can be bounded by*

$$\sum_{l=1}^{\Delta} \binom{n}{l} (q-1)^l \leq 2^{-n\left[\frac{\Delta}{n}\log(\frac{1}{q-1}) - H(\frac{\Delta}{n})\right]}, \tag{A.2}$$

where $H(x)$ represents the binary entropy function; i.e.,$H(x) = -x\log x - (1-x)\log(1-x)$, where log *represent binary logarithm.*

Proof: Denote $S \triangleq \sum_{l=1}^{\Delta} \binom{n}{l}(q-1)^l$. Consider

$$\frac{1}{q^n} S = \frac{1}{q^n} \sum_{l=1}^{\Delta} \binom{n}{l} (q-1)^l = \sum_{l=1}^{\Delta} \binom{n}{l} (\frac{1}{q})^{(n-l)} (\frac{q-1}{q})^l. \tag{A.3}$$

Let T denote a random variable taking on values in $\{0, 1\}$ with $\text{Prob}(T = 0) = 1/q$ and $\text{Prob}(T = 1) = (q-1)/q$. Therefore, (A.3) is the probability that the sum of a series of n random variables $(T_1, T_2, \ldots, T_n)$ is smaller than or equal to Δ, i.e., $\text{Prob}(\sum_{i=1}^{n} T_i \leq \Delta)$. In (Wozencraft and Reiffen, 1961; Wozencraft and Jacobs, 1965), this may be upper bounded using the Chernoff bound, yielding

$$\begin{aligned} \text{Prob}(\sum_{i=1}^{n} T_i \leq \Delta) &\leq 2^{-n\left[H(\text{Prob}(T=1)) - H(\frac{\Delta}{n}) + (\frac{\Delta}{n} - \text{Prob}(T=1))\log\frac{\text{Prob}(T=0)}{\text{Prob}(T=1)}\right]} \\ &= 2^{-n\left[H(\frac{q-1}{q}) - H(\frac{\Delta}{n}) + (\frac{\Delta}{n} - \frac{q-1}{q})\log\frac{1}{q-1}\right]} \end{aligned} \tag{A.4}$$

Therefore,

$$\begin{aligned} S &= q^n \text{Prob}(\sum_{i=1}^{n} T_i \leq \Delta) \\ &\leq 2^{n\log q} \cdot 2^{-n\left[H(\frac{q-1}{q}) - H(\frac{\Delta}{n}) + (\frac{\Delta}{n} - \frac{q-1}{q})\log\frac{1}{q-1}\right]} \\ &= 2^{-n\left[H(\frac{q-1}{q}) - H(\frac{\Delta}{n}) + (\frac{\Delta}{n} - \frac{q-1}{q})\log\frac{1}{q-1} - \log q\right]}; \end{aligned} \tag{A.5}$$

we may expand $H(\frac{q-1}{q})$, and after some manipulations, (A.5) becomes

$$S \leq 2^{-n\left[\frac{\Delta}{n}\log(\frac{1}{q-1}) - H(\frac{\Delta}{n})\right]}, \tag{A.6}$$

which proves the lemma.

∎

When $q = 2$, the result of Lemma 1 becomes

$$\sum_{l=1}^{\Delta} \binom{n}{l} \leq 2^{nH(\frac{\Delta}{n})}. \tag{A.7}$$

Lemma 1 is an important result in the discussion on the bounds of minimum distances and free distances for q-ary codes, and is also useful for proving several theorems in this appendix.

We may proceed to bound $\text{Prob}(d_{\min} \leq \delta n)$ for $0 < \delta < 1$ using Lemma 1.

Theorem 1 *Over the equiprobable ensemble of q-ary parity-check codes with length n and code rate R, the probability that the minimum distance d_{min} is smaller than or equal to a given value δn, i.e., Prob$(d_{min} \leq \delta n)$ is bounded by :*

$$Prob(d_{min} \leq \delta n) \leq 2^{-n\left[(1-R)\log q + \delta \log(\frac{1}{q-1}) - H(\delta)\right]}, \tag{A.8}$$

for $0 < \delta < 1$.

Proof: $\text{Prob}(d_{\min} \leq \delta n)$ is the probability that at least one non-zero sequence of weight less than δn is a codeword, which is over bounded by the probability that all such sequences are codewords.

A randomly chosen sequence satisfies a single parity-check equation with probability $1/q$; for the linear combinations of the symbols in the sequence take on the elements in $GF(q)$ with equal probabilities. Let X denote the linear combination of a number of symbols, $\text{Prob}(X = \gamma_i) = 1/q$ for $i = 0, 1, \ldots, q-1$. Let $P(l)$ denote the probability that a sequence of weight l is a codeword; since there are $n(1-R)$ parity-check equations in a linear code,

$$P(l) = q^{-n(1-R)}, \tag{A.9}$$

regardless of l for $l \leq n$. We have

$$\text{Prob}(d_{\min} \leq \delta n) \leq \sum_{l=1}^{\delta n} \binom{n}{l} (q-1)^l P(l) = q^{-n(1-R)} \sum_{l=1}^{\delta n} \binom{n}{l} (q-1)^l. \tag{A.10}$$

(A.10) lends itself to Lemma 1. Substituting δn for Δ in Lemma 1,

$$\text{Prob}(d_{\min} \leq \delta n) \quad \leq \quad q^{-n(1-R)} \cdot 2^{-n\left[\delta \log(\frac{1}{q-1}) - H(\delta)\right]}$$

$$= \quad 2^{-n\left[(1-R)\log q+\delta\log(\frac{1}{q-1})-H(\delta)\right]}; \tag{A.11}$$

which proves the theorem.

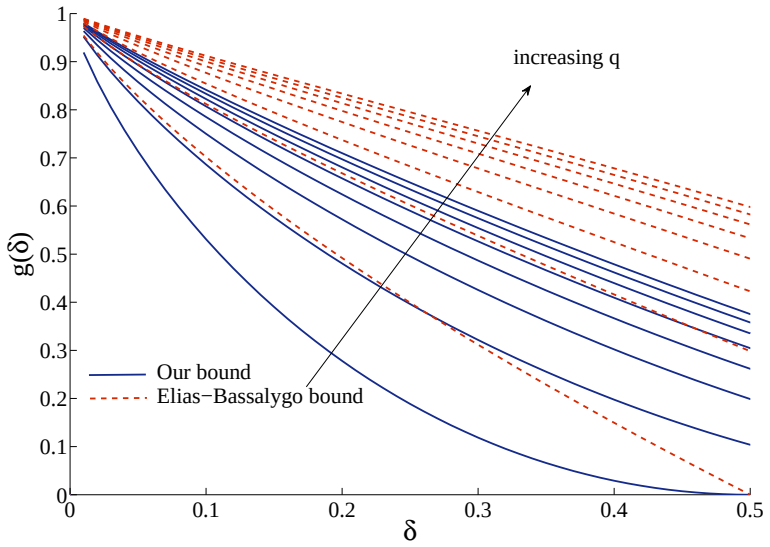


Figure A.1 $g(\delta)$ for $q = 2, 4, 8, 16, 32, 64, 128, 256$.

Theorem 1 states that the probability for the ratio of the minimum distance to the code length, i.e., $d_{\min}/n$, to be smaller than a given real number δ decreases exponentially with n if the following inequality holds :

$$(1-R)\log q + \delta\log(\frac{1}{q-1}) - H(\delta) > 0; \tag{A.12}$$

or equivalently,

$$R < 1 - \frac{H(\delta) - \delta\log(\frac{1}{q-1})}{\log q} \triangleq g(\delta); \tag{A.13}$$

i.e., $g(\delta)$ is an upper bound on the coding rate R. (A.13) is similar in form to the Elias-Bassalygo bound (Bassalygo, 1965), which applies the Johnson bound (Johnson, 1962) together with the q-ary entropy function. When $q = 2$, (A.13) is simplified as

$$R < 1 - H(\delta). \tag{A.14}$$

$g(\delta)$ is plotted in Fig. A.1 ; where it is compared to the Elias-Bassalygo bound (Bassalygo, 1965). Those values of R above these curves are undesirable, for the probability for $d_{\min}$ to

be smaller than δn grows with n. As illustrated in Fig. A.1, the 'desirable' region expands as q gets large. It is also noted that compared to the Elias-Bassalygo bound, $g(\delta)$ represents a tighter restriction on the coding rate R.

A.2 Bounds on the distances of convolutional codes

In the trellis-based decoding of convolutional codes, the free distance d_{free} is an important parameter in determining the probability that an error sequence is more likely to be received than the correct sequence of the same length, and therefore affects the possibility of confusing one codeword with another. d_{free} is the minimum weight of all non-zero codewords starting from and eventually ending in the all-zero state. In this section, we provide the upper and lower bounds of d_{free} for q-ary time invariant convolutional codes as an extension of the bounds given in (Costello, 1974). The lower bound concerns the existence of a code, in the ensemble of codes, whose free distance is above a value dependent on R and q, while the upper bound is obtained by examining the average weights of all the codewords for a specific code. Although for unterminated q-ary RCDO codes, the minimum distance $d_{\min}$ determines the pairwise error probability of decoding, the free distance d_{free} provides a more general measure of codes regardless of the implementation on the decoder.

However, before we derive a lower bound for d_{free}, we need to examine the minimum distance $d_{\min}$ of the convolutional codes.

A.2.1 A lower bound on $d_{\min}$ for convolutional codes

Denote m as the memory order of a convolutional code. Let K denote the number of q-ary information symbols entering the encoder during each time slot, the encoder generates N code symbols. The constraint length is therefore $n_A = (m+1)N$ symbols. The minimum distance is defined as the minimum weight of all non-zero codewords with length n_A generated by a sequence of $(m+1)K$ information symbols, within which the first block of K information symbols are not all 0s.

We follow Massey's rationale (Massey, 1963) in bounding $d_{\min}$: for all the convolutional codes with constraint length n_A, consider the set of non-zero sequences with weights from 1 to $(d-1)$ for $d-1 \leq (m+1)N$ symbols; if the number of codes in which these sequences are codewords is smaller than the total number of codes, there exists at least one code with $d_{\min} \geq d$.

Theorem 2 *Among all convolutional codes with code rate $R = K/N$ and memory order m, there exists at least one code with minimum distance $d_{min} \geq d$, where d is the smallest integer*

satisfying

$$\frac{d}{n_A}\log(\frac{1}{q-1}) + (1-R)\log q < H(\frac{d}{n_A}), \tag{A.15}$$

for $d \leq (m+1)N + 1$.

Proof: As in (Massey, 1963), it is most direct to calculate the number of the codes in the equiprobable ensemble through the parity-check matrices. Specifically, we consider the time-invariant codes with the following parity-check matrix :

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_0 & & & \\ \mathbf{H}_1 & \mathbf{H}_0 & & \\ \mathbf{H}_2 & \mathbf{H}_1 & \mathbf{H}_0 & \\ \vdots & \vdots & \vdots & \ddots \\ \mathbf{H}_m & \mathbf{H}_{m-1} & \mathbf{H}_{m-2} & \\ & \mathbf{H}_m & \mathbf{H}_{m-1} & \\ & & \mathbf{H}_m & \\ & & & \ddots \end{bmatrix}, \tag{A.16}$$

with each $\mathbf{H}_i, i = 0, 1, \ldots, m$ representing an $(N-K) \times N$ matrix over $GF(q)$. Let η denote the column weight of $\mathbf{H}$, i.e., η is the number of non-zero entries in each column of $\mathbf{H}$. The code is specified by N such columns; and therefore, the total number of codes is

$$\left[\binom{(m+1)(N-K)}{\eta}(q-1)^\eta\right]^N. \tag{A.17}$$

Furthermore, each sequence is checked by at most $(m+1)(N-K)$ independent parity-check equations. Hence the number of codes in which a particular sequence of length n_A is a codeword is given by

$$\frac{\left[\binom{(m+1)(N-K)}{\eta}(q-1)^\eta\right]^N}{q^{(m+1)(N-K)}}. \tag{A.18}$$

The number of non-zero sequences of length $(m+1)N$ with weight less than d is

$$\sum_{i=1}^{d-1}\binom{(m+1)N}{i}(q-1)^i = \sum_{i=1}^{d-1}\binom{n_A}{i}(q-1)^i. \tag{A.19}$$

Therefore, following the previous explanation, if

$$\frac{\left[\binom{(m+1)(N-K)}{\eta}(q-1)^{\eta}\right]^{N}}{q^{(m+1)(N-K)}}\cdot\sum_{i=1}^{d-1}\binom{n_A}{i}(q-1)^{i}<\left[\binom{(m+1)(N-K)}{\eta}(q-1)^{\eta}\right]^{N}, \quad \text{(A.20)}$$

or equivalently, if

$$\frac{\sum_{i=1}^{d-1}\binom{n_A}{i}(q-1)^{i}}{q^{(m+1)(N-K)}}<1, \quad \text{(A.21)}$$

there exists at least one code with $d_{\min}\geq d$. Again, we may bound the numerator on the left hand side using Lemma 1, leading to

$$\begin{aligned}
2^{-n_A\left[\frac{d-1}{n_A}\log(\frac{1}{q-1})-H(\frac{d-1}{n_A})\right]}\cdot q^{-(m+1)(N-K)} &< 1\\
2^{-n_A\left[\frac{d-1}{n_A}\log(\frac{1}{q-1})-H(\frac{d-1}{n_A})\right]}\cdot 2^{-(1-R)n_A\log q} &< 1\\
2^{-n_A\left[\frac{d-1}{n_A}\log(\frac{1}{q-1})-H(\frac{d-1}{n_A})+(1-R)\log q\right]} &< 1\\
\frac{d-1}{n_A}\log(\frac{1}{q-1})-H(\frac{d-1}{n_A})+(1-R)\log q &> 0. \quad \text{(A.22)}
\end{aligned}$$

Finding the largest integer d satisfying (A.22) is equivalent to finding the smallest d such that

$$\frac{d}{n_A}\log(\frac{1}{q-1})+(1-R)\log q<H(\frac{d}{n_A}), \quad \text{(A.23)}$$

for $d\leq(m+1)N+1$.

∎

Note that although we consider the codes with fixed column weight (η) in the proof, the result applies to codes with uneven weight distribution as well. If $q=2$, i.e., if we are considering binary codes, the condition in (A.15) is reduced to the simpler form as in (Massey, 1963) :

$$1-R<H(\frac{d}{n_A}); \quad \text{(A.24)}$$

or equivalently,

$$\frac{d}{n_A}>H^{-1}(1-R). \quad \text{(A.25)}$$

Furthermore, we are concerned whether there exists a d that satisfies (A.15). Let $t\triangleq d/n_A$, and define a function

$$f(t)=\log(\frac{1}{q-1})t+(1-R)\log q-H(t), \quad \text{(A.26)}$$

with $0\leq t<1$. Note that $H(t)$ is the binary entropy function, and $\log(\frac{1}{q-1})t+(1-R)\log q$

is a line with intercept $(1-R)\log q > 0$ and slope $\log(\frac{1}{q-1}) < 0$ for $q > 2$. Setting the first derivative of $f(t)$ to 0 :

$$\begin{aligned} f'(t) = \log(\frac{1}{q-1}) + \log(\frac{t}{1-t}) &= 0 \\ t &= \frac{q-1}{q}. \end{aligned} \tag{A.27}$$

Substituting $t = (q-1)/q$ into $f(t)$ we have

$$\begin{aligned} f(\frac{q-1}{q}) &= \frac{q-1}{q}\log(\frac{1}{q-1}) + (1-R)\log q + \frac{q-1}{q}\log(\frac{q-1}{q}) + \frac{1}{q}\log(\frac{1}{q}) \\ &= -R\log q < 0, \end{aligned} \tag{A.28}$$

for $q > 2$.

Hence, for $q > 2$, $f(0) = (1-R)\log q > 0$ and $f((q-1)/q) = -R\log q < 0$; moreover, since both $\log(\frac{1}{q-1})t + (1-R)\log q$ and $H(t)$ are continuous functions for $0 \le t < 0$, $f(t)$ is also a continuous function in this range. Therefore, there exists a $t' \in (0, (q-1)/q)$ satisfying $f(t') = 0$ regardless of the value of R.

When $q = 2$, $f(t) = 1 - R - H(t)$; since the value of $-H(t)$ ranges from -1 to 0 and $1 - R > 0$, there also exists a t' such that $f(t') = 0$.

In summary, a d satisfying (A.15) exists for $q \ge 2$; and we denote it as d_l in our following discussion.

A.2.2 A lower bound on d_{free}

In this section, we present a lower bound on the free distance d_{free} of convolutional codes. Since d_{free} is the minimum weight of all those codewords starting from and ending in the all-zero state, unlike for $d_{\min}$, we need to consider information sequences of different lengths instead of directly discussing the encoded sequences. Denote m as the memory order to convolutional codes; i.e., m is maximum number of delayed time slots in the shift registers of convolutional codes. Let $\mathbf{x}_t$ denote K information symbols at time t, $\mathbf{x}_t \in [GF(q)]^K$. For an information sequence of T time slots, i.e., $(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{T-1})$ with $\mathbf{x}_0 \neq \mathbf{0}$ and $\mathbf{x}_{T-1} \neq \mathbf{0}$, starting from the $T^{\,th}$ time slot, if there is no further input sequence, it takes the encoder *at most* another T time slot to return to the all-zero state. Specifically, if the length of the information sequence is longer than the constraint length n_A, it takes the encoder at most m time slots to return to the all-zero state by appending mK 0s to the information sequence; otherwise, it takes the encoder at most T time slots to return to the all-zero state due to the symmetry property of the underlying trellis of a convolutional code. Using the rationale in

(Costello, 1974), these two scenarios are considered separately.

(a) $T > m + 2$

As mentioned earlier, m additional state transitions are needed to guarantee that the encoder return to the all-zero state. In this case, the code sequence contains more than $2n_A$ possibly non-zero code symbols. However, if the first and the last m state transitions in the entire code sequences are symmetrical, the $(T - m)$ state transitions in the middle of the sequence potentially increase the weight of the codeword by inducing state transitions with output sequence with non-zero weight. Therefore, for $T > m + 2$, the smallest weight of a codeword starting from and ending the all-zero state is greater than the double of the bound on $d_{\min}$, i.e., $2d_l$ as given in Theorem 2.

(b) $T \leq m + 2$

Under this scenario, the potentially non-zero symbols span less than two constraint length $(2n_A)$. The encoder needs only another $\min(T, m)$ blocks after the T information block to return to the all-zero state. However, it is nontrivial to determine the value of T that would provide the codeword with the smallest weight. Therefore, we may apply similar rationale as in Theorem 2 in bounding the distance and then optimize the bound with respect to T. Note that unlike in Theorem 2, we need to consider information sequences rather than code sequences directly.

Theorem 3 *From all the time-invariant convolutional codes with code rate R defined over $GF(q)$, there exists at least one code with free distance d_{free} satisfies the following inequalities :*

- *If one of the following 2 conditions is satisfied :*
  1. $R \geq \frac{1}{2}$;
  2. $R < \frac{1}{2}$ *and* $\frac{R \log q}{(1-q^{2R-1})[\log[(q-1)q^{2R-1}] - \log(1-q^{2R-1})]} > 1$

  the free distances of the code satisfies

$$\lim_{m \to \infty} \frac{d_{free}}{n_A} \geq 2\delta_l, \tag{A.29}$$

  where $\delta_l = d_l/n_A$, and d_l is the smallest integer satisfying (A.15) ;
- *otherwise,*

$$\lim_{m \to \infty} \frac{d_{free}}{n_A} \geq 2 \min(\delta_l, \delta_s); \tag{A.30}$$

  where

$$\delta_s = \frac{R \log q}{\log[(q-1)q^{2R-1}] - \log(1 - q^{2R-1})}. \tag{A.31}$$

Proof: For $T > m+2$, we have concluded that the minimum distance for codewords departing from and reverting to the all-zero state is bounded by $2d_l$; in the proof, we concentrate on the case when $T \leq m+2$.

We may follow the same procedure as in (Costello, 1974). Consider the information sequence $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{T-1})$ followed by mK 0s for $T \leq m+2$. Since the first and second half of the state transitions are symmetrical, we may bound them with the same procedure.

The first half contains $L = (T+m)/2 \leq (m+1)$ information blocks. Consider the encoding equation

$$\mathbf{y} = \mathbf{G}\mathbf{x}, \tag{A.32}$$

where $\mathbf{y}$ and $\mathbf{G}$ represents the generated code sequence and the generator matrix respectively. For a time-invariant convolutional code, the $\mathbf{G}$ takes the form

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \mathbf{G}_2 & \ldots & \mathbf{G}_m & & & \\ & \mathbf{G}_0 & \mathbf{G}_1 & \ldots & \mathbf{G}_{m-1} & \mathbf{G}_m & & \\ & & \mathbf{G}_0 & \ldots & \mathbf{G}_{m-2} & \mathbf{G}_{m-1} & \mathbf{G}_m & \\ & & & \ddots & & & & \ddots \end{bmatrix}, \tag{A.33}$$

A convolutional code is fully specified by its generator matrix $\mathbf{G}$, therefore the number of different $\mathbf{G}$s is the same as the number of parity-check matrices $\mathbf{H}$ as discussed previously, i.e., $\left[\binom{(m+1)(N-K)}{\eta}(q-1)^\eta\right]^N$. Moreover, (A.32) represents a set of LN linear equations ; therefore the number of codes satisfying (A.32) *for a particular information sequence* is

$$\frac{\left[\binom{(m+1)(N-K)}{\eta}(q-1)^\eta\right]^N}{q^{NL}}. \tag{A.34}$$

Hence, the total number of codes giving rise to code sequence of weight ranging from 1 to $(d-1)$ for this information sequence is given by

$$\frac{\left[\binom{(m+1)(N-K)}{\eta}(q-1)^\eta\right]^N \cdot \sum_{i=1}^{d-1}\binom{NL}{i}(q-1)^i}{q^{NL}}. \tag{A.35}$$

Therefore, the number of different codes resulting in those codewords whose weight in the first half ranging from 1 to $(d-1)$ for any information sequence spanning *at most* $(m+2)$ blocks is given by

$$\sum_{T=1}^{m+2} \frac{q^{TK}\left[\binom{(m+1)(N-K)}{\eta}(q-1)^\eta\right]^N \cdot \sum_{i=1}^{d-1}\binom{NL}{i}(q-1)^i}{q^{NL}}. \tag{A.36}$$

Note again the information blocks satisfies the restriction that $\mathbf{x}_0 \neq \mathbf{0}$ and $\mathbf{x}_{T-1} \neq \mathbf{0}$. The same calculation could be repeated for the latter half of the code sequence.

Similar to the reasoning in (Costello, 1974), if the number of codes giving rise to those codewords with weight smaller than d in either its first or second half is less than the total number of the codes, there must be at least one code with minimum codeword weight $2d$ for $T \leq m+2$, which is represented by the following condition :

$$\begin{aligned} 2\sum_{T=1}^{m+2} \frac{q^{TK}\left[\binom{(m+1)(N-K)}{\eta}(q-1)^{\eta}\right]^{N} \cdot \sum_{i=1}^{d-1}\binom{NL}{i}(q-1)^{i}}{q^{NL}} &< \left[\binom{(m+1)(N-K)}{\eta}(q-1)^{\eta}\right]^{N}, \\ 2\sum_{T=1}^{m+2} \frac{q^{TK}\sum_{i=1}^{d-1}\binom{NL}{i}(q-1)^{i}}{q^{NL}} &< 1, \end{aligned} \tag{A.37}$$

where $L = (T+m)/2$.

Again, we may apply Lemma 1 to simplify (A.37),

$$\begin{aligned} & 2\sum_{T=1}^{m+2} \frac{q^{TK}\sum_{i=1}^{d-1}\binom{NL}{i}(q-1)^{i}}{q^{NL}} \\ \leq\ & 2\sum_{T=1}^{m+2} q^{TK-NL} \cdot 2^{-NL[\frac{d-1}{NL}\log(\frac{1}{q-1})-H(\frac{d-1}{NL})]} \\ =\ & 2\sum_{T=1}^{m+2} 2^{RNT\log q - NL[\frac{d-1}{NL}\log(\frac{1}{q-1})-H(\frac{d-1}{NL})+\log q]} < 1. \end{aligned} \tag{A.38}$$

We intend to bound the summation of $(m+2)$ terms in (A.38) by their largest terms, i.e., we need to find the T that maximizes

$$RNT\log q - NL[\frac{d-1}{NL}\log(\frac{1}{q-1}) - H(\frac{d-1}{NL}) + \log q] \tag{A.39}$$

denoted as T_{max}, and therefore bounding (A.38) with

$$2(m+2)2^{RNT_{max}\log q - NL_{max}[\frac{d-1}{NL_{max}}\log(\frac{1}{q-1})-H(\frac{d-1}{NL_{max}})+\log q]} < 1, \tag{A.40}$$

where $L_{max} = (T_{max}+m)/2$.

The first partial derivative of (A.39) with respect to T is given by

$$\begin{aligned} RN\log q - \frac{N}{2}\left[\log q - H(\frac{d-1}{NL}) + \frac{d-1}{NL}\log(\frac{1}{q-1})\right] \\ + \frac{N}{2}\left[\frac{d-1}{NL}\log(\frac{\frac{d-1}{NL}}{1-\frac{d-1}{NL}}) + \frac{d-1}{NL}\log(\frac{1}{q-1})\right] \end{aligned}$$

$$= \quad RN\log q - \frac{N}{2}\left[\log q + \log(1-\frac{d-1}{NL})\right]; \tag{A.41}$$

and the second partial derivative is given by

$$-\frac{N}{2}\left[\frac{\log e}{1-\frac{d-1}{NL}}(-\frac{d-1}{N})(-\frac{1}{L^2})(\frac{1}{2})\right] = -\frac{N\log e}{4L}\cdot\frac{1}{\frac{NL}{d-1}-1}, \tag{A.42}$$

where e denotes the Euler's number. For $d-1 < NL$, the second partial derivative is always negative, indicating the possibility of a local maxima. Setting the first partial to be greater than 0 :

$$\begin{aligned} RN\log q - \frac{N}{2}\left[\log q + \log(1-\frac{d-1}{NL})\right] &> 0, \\ R &> \frac{1}{2}\left[1+\frac{\log(1-\frac{d-1}{NL})}{\log q}\right] \\ &. \end{aligned} \tag{A.43}$$

Note that the right hand side of (A.43) is smaller than 1/2, therefore, the first partial is always positive when $R \geq 1/2$, i.e., (A.39) is monotonically increasing with T. Hence we may substitute $T = m+2$ into (A.39) to bound the lefthand side of (A.38), leading to

$$2(m+2)\cdot 2^{RN(m+2)\log q - n_A[\log q - H(\frac{d-1}{n_A}) + \frac{d-1}{n_A}\log(\frac{1}{q-1})]} < 1,$$

or

$$RN(m+2)\log q - n_A[\log q - H(\frac{d-1}{n_A}) + \frac{d-1}{n_A}\log(\frac{1}{q-1})] < \log[\frac{1}{2(m+2)}]. \tag{A.44}$$

Dividing both sides of (A.44) by m and let m approach ∞, we have

$$H(\frac{d-1}{n_A}) < (1-R)\log q + \frac{d-1}{n_A}\log(\frac{1}{q-1}); \tag{A.45}$$

Seeking the largest d satisfying the above equation is equivalent to seeking the smallest d satisfying (A.15), therefore, the minimum distance of the first half of the codewords is bounded by d_l. Since the same analysis could be applied on the second half, we conclude that for $R \geq 1/2$ and $T < (m+2)$, there exists a code whose minimum weight of these codewords are lower bounded by $2d_l$.

When $R < 1/2$, the first partial derivative could be negative, setting it to 0, we may find

a local maxima of (A.39) :

$$\frac{d-1}{NL_{max}} = 1-q^{2R-1}, \tag{A.46}$$

$$T_{max} = \frac{2(d-1)}{N(1-q^{2R-1})} - m. \tag{A.47}$$

However, it is not guaranteed that the value in (A.47) is within the range $T \leq m+2$; furthermore, the value of T_{max} is still dependent on d. Recall that our goal is to seek the largest d satisfying (A.38). Since the right hand side of (A.46) is a constant, seeking the largest d is equivalent to seeking the largest L, and therefore the largest T under the condition of (A.46). We may substitute (A.46) into (A.40) and let $m \to \infty$:

$$R\frac{T_{max}}{L_{max}}\log q < \log q - H(1-q^{2R-1}) + (1-q^{2R-1})\log(\frac{1}{q-1}),$$

or

$$\begin{aligned}\frac{m+T_{max}}{2m} &< \frac{R\log q}{H(1-q^{2R-1}) - (1-q^{2R-1})\log(\frac{1}{q-1}) - (1-2R)\log q} \\ &= \frac{R\log q}{(1-q^{2R-1})[\log[(q-1)q^{2R-1}] - \log(1-q^{2R-1})]}.\end{aligned} \tag{A.48}$$

When the right hand side of (A.48) is greater than 1, the largest T is $(m+2)$, i.e., $T_{max} = m+2$, and therefore providing the same bound as in the case when $R \geq 1/2$, i.e., $2d_l$.

However, when

$$\frac{R\log q}{(1-q^{2R-1})[\log[(q-1)q^{2R-1}] - \log(1-q^{2R-1})]} \leq 1, \tag{A.49}$$

we may not find the a T_{max} independent of d. Substituting (A.47) directly into (A.40) and letting $m \to \infty$,

$$\begin{aligned}&\frac{RN}{m}\log q\left[\frac{2(d-1)}{N(1-q^{2R-1})} - m\right] \\ &\qquad -\frac{N}{m}\cdot\frac{d-1}{N(1-q^{2R-1})}\left[\log q - H(1-q^{2R-1}) + (1-q^{2R-1})\log(\frac{1}{q-1})\right] < 0;\end{aligned} \tag{A.50}$$

after some manipulations,

$$\frac{d-1}{n_A} < \frac{R\log q}{\log[(q-1)q^{2R-1}] - \log(1-q^{2R-1})}. \tag{A.51}$$

Again, seeking the largest d satisfying the above equation is equivalent to seeking the smallest d such that

$$\frac{d}{n_A} \geq \frac{R \log q}{\log[(q-1)q^{2R-1}] - \log(1-q^{2R-1})} = \delta_s. \tag{A.52}$$

Furthermore, due to the symmetry property of the convolutional codes, the same bound applies to the second half of the codewords as well.

In summary,

1. For $T > m+2$, there exists a code whose minimum weight of codewords departing from and reverting to the all-zero state is lower bounded by $2d_l$;
2. For $T \leq m+2$, and either one of the following conditions is satisfied :
   (a) $R \geq \frac{1}{2}$;
   (b) $R < \frac{1}{2}$ and $\frac{R \log q}{(1-q^{2R-1})[\log[(q-1)q^{2R-1}] - \log(1-q^{2R-1})]} > 1$;
   there exists a code with the same bound as above.
3. For $T \leq m+2$, and neither of the two conditions above is satisfied, there exists a code whose minimum weight is lowered bounded by $\delta_s n_A$.

Combining the above results, we come up the lower bound on d_{free} independent of T as in the theorem.

∎

Note that the bound in Theorem 3 can be reduced to the bound given by Costello (Costello, 1974) when $q = 2$, as described in (2.14), which is shown in the following.

When $q = 2$, the condition $\frac{R \log q}{(1-q^{2R-1})[\log[(q-1)q^{2R-1}] - \log(1-q^{2R-1})]} > 1$ is reduced to

$$\frac{R}{(1-2^{2R-1})[(2R-1) - \log(1-2^{2R-1})]} > 1, \tag{A.53}$$

or equivalently $R > 0.379$. Furthermore, as $q = 2$, d_l is the smallest integer satisfying (A.25) ; i.e., $d_l/n_A = H^{-1}(1-R)$. Therefore, the first case in the theorem becomes : when $R > 0.378$,

$$\lim_{m \to \infty} \frac{d_{\text{free}}}{n_A} \geq 2H^{-1}(1-R). \tag{A.54}$$

For the second case in the theorem, i.e., $R \leq 0.378$, when $q = 2$,

$$\begin{aligned} \delta_s &= \frac{R}{2R - 1 - \log(1-2^{2R-1})} \\ &= \frac{R(1-2^{2R-1})}{(2R-1)(1-2^{2R-1}) - (1-2^{2r-1})\log(1-2^{2R-1})} \end{aligned}$$

$$
\begin{aligned}
&= \frac{R(1-2^{2R-1})}{(2R-1)-(2R-1)2^{2R-1}-(1-2^{2R-1})\log(1-2^{2R-1})} \\
&= \frac{R(1-2^{2R-1})}{(2R-1)-2^{2R-1}\log(2^{2R-1})-(1-2^{2R-1})\log(1-2^{2R-1})} \\
&= \frac{R(1-2^{2R-1})}{(2R-1)+H(2^{2R-1})}. \qquad \text{(A.55)}
\end{aligned}
$$

Furthermore, for $R \leq 0.378$, it can be numerically verified that $\delta_s < \delta_l$. Therefore, the inequalities in the theorem becomes

$$
\lim_{m\to\infty} \frac{d_{\text{free}}}{n_A} \geq \begin{cases} 2H^{-1}(1-R), & 0.378 \leq R \leq 1; \\ \frac{2R(1-2^{2R-1})}{H(2^{2R-1})+2R-1}, & 0 \leq R \leq 0.378; \end{cases} \qquad \text{(A.56)}
$$

which is exactly the same as (2.14).

A.2.3 An upper bound on d_{free}

The main idea of upper bounding the free distance d_{free} of convolutional codes follows that used in the Plotkin bound (Plotkin, 1960), which had also been applied in bounding the minimum distance and free distance of binary convolutional codes (Massey, 1963; Costello, 1974) : *the free distance of the convolutional codes is upper bounded by the average weight of non-zero codewords.*

Following the definition in (Costello, 1974), define the memory span of the $j^{\,th}$ transmitted sequence to be

$$
M_j = \max_{0\leq i\leq m}[i|\text{the } j^{\,th} \text{ column of } \mathbf{G}_i \neq \mathbf{0}], \qquad \text{(A.57)}
$$

for $j = 1, 2, \ldots, N$. The effective constraint length of the convolutional codes is defined as

$$
n_E = \sum_{j=1}^{N} M_j + N, \qquad \text{(A.58)}
$$

which is the number of symbols affected by a single non-zero information block of K symbols. Note that $n_E \leq n_A$.

Theorem 4 *The free distance of any time-invariant convolutional code defined over $GF(q)$ satisfies*

$$
d_{free} \leq \frac{q-1}{q}\left[\frac{\log[(q-1)n_E]}{R\log q} + n_E\right] + 1 \qquad \text{(A.59)}
$$

if

$$
\frac{(q-1)n_E}{\log[(q-1)n_E] - \log q + K} > \frac{1}{R}. \qquad \text{(A.60)}
$$

Proof: Again, we may consider an information sequence of T blocks with $\mathbf{x}_0 \neq \mathbf{0}$ and $\mathbf{x}_{T-1} \neq \mathbf{0}$; the number of possibly non-zero code symbols is therefore $NT + \sum_{j=1}^{N} M_j$. Since the probability for a single code symbol to be non-zero is $(q-1)/q$, the average weight of the code sequence is given by

$$\left(NT + \sum_{j=1}^{N} M_j \right) \cdot \frac{q-1}{q} = \frac{q-1}{q}(NT + n_E - N). \tag{A.61}$$

However, (A.61) actually concerns the average weight of a code sequence generated by a random information sequence; but the all-zero information sequence (and hence the all-zero code sequence) should be excluded from our consideration. Hence, d_{free} is bounded by the average weight of code sequence generated by all non-zero information sequences, leading to

$$\begin{aligned} d_{\text{free}} &\leq \frac{q^{KT}}{q^{KT}-1} \cdot \frac{q-1}{q}(NT + n_E - N) \\ &= \left(1 + \frac{1}{q^{KT}-1}\right) \cdot \frac{q-1}{q}(NT + n_E - N) \\ &= \frac{q-1}{q}(NT + n_E - N) + \frac{1}{q^{KT}-1} \cdot \frac{q-1}{q}(NT + n_E - N), \end{aligned} \tag{A.62}$$

which is still dependent on T.

The procedure for bounding (A.62) is similar to that in (Costello, 1974), which can be more directly summarized as :

Since

$$\lim_{T\to\infty} \frac{1}{q^{KT}-1} \cdot \frac{q-1}{q}(NT + n_E - N) = 0, \tag{A.63}$$

the second term in (A.62) *can be bounded by some constant when T is large enough. On the other hand, the first term grows linearly with T; therefore, we are to find such a T that is large enough to bound the second term but still not too large to bound the first term.*

We may actually pick any constant to bound the second term. Here we pick 1, i.e., our goal is to find T such that

$$\begin{aligned} \frac{1}{q^{KT}-1} \cdot \frac{q-1}{q}(NT + n_E - N) &< 1 \\ \frac{q-1}{q}NT + \frac{q-1}{q}n_E - \frac{q-1}{q}N &< q^{KT} - 1. \end{aligned} \tag{A.64}$$

When $N \geq 2$, the term $-N(q-1)/q$ is smaller than -1 for all values of q. Therefore, we may

loosen the restriction to be

$$\frac{q-1}{q}NT + \frac{q-1}{q}n_E < q^{KT}. \tag{A.65}$$

Select T satisfying

$$\frac{1}{q-1}q^{K(T-1)} < n_E < \frac{1}{q-1}q^{KT}, \tag{A.66}$$

where the right hand side is to bound n_E with T and the left hand side guarantees that KT (NT) is still bounded. The lefthand side of (A.66)can be modified into

$$KT < \frac{\log[(q-1)n_E]}{\log q} + K \tag{A.67}$$

where $K = RN$. With the right hand side of (A.66), we have

$$\frac{q-1}{q}n_E < \frac{1}{q}q^{KT}. \tag{A.68}$$

From (A.66), since $(q-1)/q + 1/q = 1$, it is obvious that we need the inequality $NT < q^{KT}$ for (A.65) to be satisfied. Moreover, from the right hand side of (A.66), we know that $(q-1)n_E < q^{KT}$, and in (A.67), we have bounded KT ; therefore, we may set the requirement

$$NT < \frac{1}{R}\left[\frac{\log[(q-1)n_E]}{\log q} + K\right] < (q-1)n_E < q^{KT}. \tag{A.69}$$

After some manipulation, the inequality in the middle of (A.69) becomes

$$\frac{(q-1)n_E}{\log[(q-1)n_E] - \log q + K} > \frac{1}{R}, \tag{A.70}$$

which is the condition in the theorem.

In summary, we may substitute the right hand size of (A.64) together with (A.69) into (A.62), resulting in

$$\begin{aligned} d_{\text{free}} &< \frac{q-1}{q}(NT + n_E - N) + \frac{1}{q^{KT}-1}\cdot\frac{q-1}{q}(q^{KT} + \frac{1}{q-1}q^{KT} - N) \\ &= \frac{q-1}{q}(NT + n_E - N) + \frac{1}{q^{KT}-1}(q^{KT} - \frac{q-1}{q}N) \\ &< \frac{q-1}{q}(NT + n_E - N) + \frac{1}{q^{KT}-1}(q^{KT} - 1) \\ &= \frac{q-1}{q}(NT + n_E - N) + 1. \end{aligned} \tag{A.71}$$

$$\tag{A.72}$$

Furthermore, substituting (A.67) into the above inequality,

$$\begin{aligned} d_{\text{free}} &< \frac{q-1}{q}\left\{\frac{1}{R}\left[\frac{\log[(q-1)n_E]}{\log q}+K\right]+n_E-N\right\}+1 \\ &= \frac{q-1}{q}\left[\frac{\log[(q-1)n_E]}{R\log q}+n_E\right]+1, \end{aligned} \tag{A.73}$$

proving the theorem.

∎

A few notes need to be made on this theorem :

1. Since we have used n_A in the lowering bounding the free distances d_{free}, we may also apply n_A in the upper bound instead of n_E ; however, this leads to a looser bound.
2. The approach in bounding the second term in (A.62) can be replaced with other methods in which we may transform the terms in the parenthesis into a form similar to $(q^{KT}-1)$, leading to slightly different bounds.
3. The restriction in the theorem is naturally satisfied when $m \to \infty$, for $n_E \to \infty$ and therefore the lefthand side approaches infinity ; dividing both sides of (A.59) and letting $m \to \infty$, we have the following more general form

$$\lim_{m\to\infty}\frac{d_{\text{free}}}{n_E} < \frac{q-1}{q}; \tag{A.74}$$

which actually states that the bound approaches the average weight of a codeword induced by K information symbols (1 block) when m is large enough.

A.3 Numerical results

This section presents the upper and lower bounds evaluated using the results in Section A.2 and explores their implication on the error performances of the recursive convolutional doubly orthogonal (RCDO) codes defined over $GF(q)$.

Using Theorem 3, we plot the lower bound on d_{free}/n_A as a function of the coding rate R in Fig. A.2 for various q values. Note that the lower bound defined here is not the minimum value of d_{free}/n_A for a particular code with value of R and q, rather it confirms the existence of a 'good' code, i.e., from all the codes with the same parameters (R, q), there *exists* at least one code with d_{free}/n_A satisfying the bound ; while a randomly selected code from the set of all the codes may provide d_{free}/n_A below these curves. Furthermore, as mentioned earlier, the upper bound in Theorem 4 can also be derived using n_A instead of n_E as in the lower bounds, leading to a looser bound. For fair comparison with the lower bounds, in Fig. A.2, the upper

bounds on d_{free}/n_A are plotted instead of d_{free}/n_E for various q values. It is observed from the lower bounds that the benefit on the free distance by shifting to finite field with large q values decreases with increasing q, e.g., the increase in the free distance by shifting from $GF(8)$ to $GF(16)$ is less substantial than that of the shift from $GF(4)$ to $GF(8)$. Note that as mentioned in previous sections, our bounds are reduce to Costello's bounds (Costello, 1974) when $q = 2$; therefore, the curves corresponding to $q = 2$ in Fig. A.2 are the same as in (Costello, 1974).

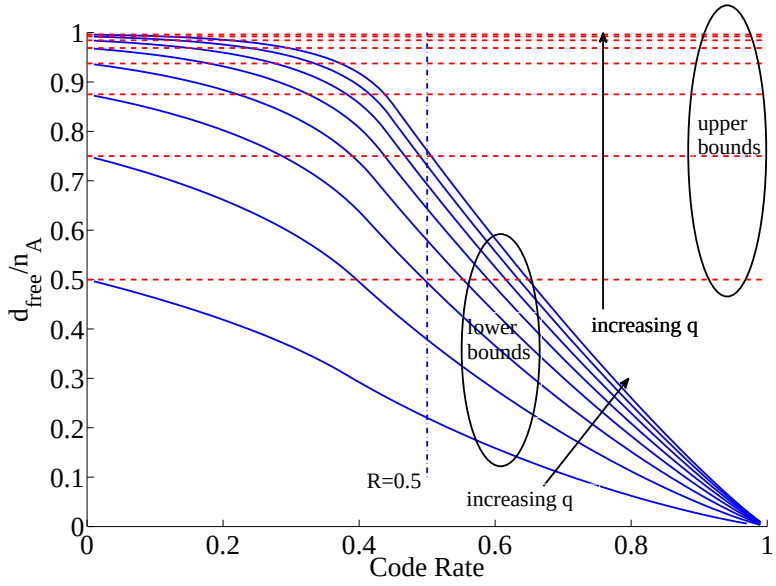


Figure A.2 Lower and upper bounds on $\frac{d_{\text{free}}}{n_A}$ as a function of R for $q = 2$, 4, 8, 16, 32, 64, 128, 256.

In the study for q-ary RCDO codes, we have examined error performances of various codes with $R = 1/2$ with different q and K values. The memory order m of these codes are listed in Table A.1. Using Theorem 3 and Theorem 4, we calculated the lower and upper bounds on the d_{free} as shown in Table A.2. Note again that the lower bounds concerns the existence of a 'good' (in terms of d_{free}) code. For instance, as shown in the table, there exists at least one 4-ary code with $K/N = 4/8$ and $R = 1/2$ whose d_{free} is greater than 104 but smaller than 168; however, it is still probable for a randomly selected 4-ary code with the same K/N and R values to have a d_{free} smaller than 104. Therefore, although the values of lower bounds are large for some codes, it is not guaranteed that a typical code may demonstrate error performances as good as the distance bounds indicate.

However, the lower bound on d_{free} may still provide some implication on the error performance differences among the codes with different q values. Consider the maximum likelihood

Table A.1 Memory order of various codes. b/c : code rate ; m : memory order

b/c	4/8	5/10	6/12	8/16	10/20	15/30
m	33	33	34	40	49	149

Table A.2 Lower and upper bounds (a, b) on d_{free} for RCDO codes with various q and K values for $R = 1/2$.

K/N	4/8	5/10	6/12	8/16	10/20	15/30
$q = 2$	(60, 146)	(75, 180)	(93, 220)	(145, 339)	(221, 511)	(991, 2264)
$q = 4$	(104, 168)	(130, 200)	(160, 237)	(250, 354)	(380, 561)	(1704, 2460)
$q = 8$	(136, 195)	(168, 233)	(208, 277)	(324, 413)	(494, 654)	(2224, 2870)
$q = 16$	(158, 210)	(198, 250)	(244, 297)	(380, 443)	(580, 701)	(2608, 3075)

decoding and assume all-zero codewords are transmitted. For large E_{b}/N_0 value, the event error probabilities are dominated by the codeword with distance d_{free} from the all-zero codeword. For codes defined over $GF(q = 2^z)$, denote the transmitted signal sequence for the all-zero codeword as $\mathbf{v}$, the signal sequence for a codeword with weight d_{free} as $\mathbf{v}'$ and the received signal sequence as $\mathbf{r}$, the *event error probability* between the two codewords is therefore

$$P_e = \text{Prob}(\mathbf{r} \cdot \mathbf{v}' - \mathbf{r} \cdot \mathbf{v} > 0). \tag{A.75}$$

Note that $\mathbf{v}$ and $\mathbf{v}'$ differs only in the positions where their corresponding code symbols differ, without loss of generality, we may assume the indices of these symbols are $1, 2, \ldots, d_{\text{free}}$, therefore

$$P_e = \text{Prob}(\sum_{i=1}^{d_{\text{free}}} r_i \cdot (v_i' - v_i) > 0). \tag{A.76}$$

Furthermore, under the binary phase shift keying (BPSK) modulation where a bit b is transmitted as $2b - 1$, each r_i contains z BPSK signals, denoted as $(r_i(1), r_i(2), \ldots, r_i(z))$, and similar notation applies to v_i and v_i', leading to

$$\begin{aligned} P_e &= \text{Prob}(\sum_{i=1}^{d_{\text{free}}} (r_i(1), r_i(2), \ldots, r_i(z)) \cdot (v_i'(1) - v_i(1), v_i'(2) - v_i(2), \ldots, v_i'(z) - v_i(z)) > 0) \\ &= \text{Prob}(\sum_{i=1}^{d_{\text{free}}} \sum_{j=1}^{z} r_i(j)(v_i'(j) - v_i(j)) > 0). \end{aligned} \tag{A.77}$$

Note that when the $j^{\,th}$ bit in the symbol v_i' is 0, $v_i'(j) - v_i(j) = 0$; otherwise $v_i'(j) - v_i(j) = 2$. Denote $\bar{u}_q$ as the average number of 1s in the binary representation of the non-zero elements

in $GF(q)$ and $d_{q,free}$ as the free distance for codes defined over $GF(q)$, (A.77) represents the probability that the summation of a set of $\bar{u}_q d_{q,free}$ random variables, each normally distributed with mean -1 and variance $N_0/2E_s$, is greater than 0 ; i.e., the summation is normally distributed with mean $-\bar{u}_q d_{q,free}$ and variance $\bar{u}_q d_{q,free} N_0/2E_s$, after some calculation, we have

$$P_e = Q\left(\sqrt{\frac{2\bar{u}_q d_{q,free} R E_b}{N_0}}\right) \approx \frac{1}{2} e^{-\frac{\bar{u}_q d_{q,free} R E_b}{N_0}}. \tag{A.78}$$

(A.78) addresses the probability that at any given time, a path diverging from and eventually ending in the all-zero state has higher reliability than the all-zero path in the ML decoding process. We may proceed to calculate the bound on the bit error probability using (A.78).

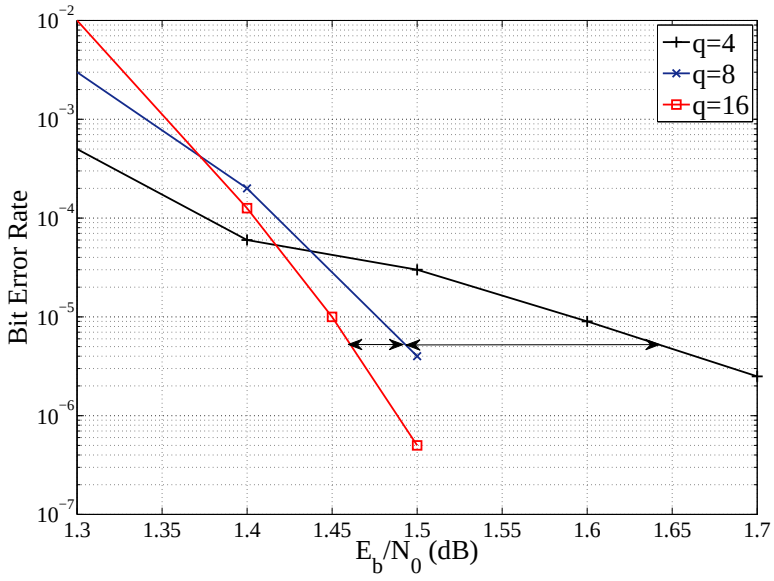


Figure A.3 Bit error rate as a function of E_b/N_0 for $R = 4/8$ RCDO codes under AWGN channel, 50 iterations.

The number of non-zero symbols in the erroneous path is d_{free}, among which Rd_{free} symbols are expected to be information symbols. As previously stated, the average number of non-zero bits in a q-ary symbol is $\bar{u}_q$. Therefore, at any give time slot, the bit error probability is calculated as

$$\begin{aligned} P_b &= \frac{1}{b}\bar{u}_q R d_{q,free} P_e \\ &\approx \frac{1}{2b}\bar{u}_q R d_{q,free} e^{-\frac{\bar{u}_q d_{q,free} R E_b}{N_0}} \end{aligned} \tag{A.79}$$

Therefore, the asymptotic gain under the maximum likelihood (ML) decoding by moving from $GF(q_1)$ to $GF(q_2)$ for the same code rate is

$$\tau = 10 \log \left(\frac{\bar{u}_{q_2} d_{q_2,free}}{\bar{u}_{q_1} d_{q_1,free}} \right); \tag{A.80}$$

where $\bar{u}_q$ can be easily calculated by enumerating the binary representation of all the non-zero elements in $GF(q)$. e.g. $\bar{u}_4 = 4/3$, $\bar{u}_8 = 12/7$, and $\bar{u}_{16} = 32/15$.

Using (A.80) and the lower bounds on the free distances given in Table A.2, we may approximate the coding gain obtained from the shift to larger alphabetical size (q). We observe that the benefit brought about through $\bar{u}_q$ by shifting to the next larger q is decreasing with q; e.g., $\bar{u}_8/\bar{u}_4 > \bar{u}_{16}/\bar{u}_8$. Furthermore, from the previous discussion, the increase in d_{free} by shifting from a q-ary finite field onto the next higher-order field is also decreasing with q. Therefore, as q gets large, it is expected that the benefits on the error performances obtained from further increasing q become less obvious, which coincides with our simulation on the error performances with RCDO codes, as demonstrated in Fig. A.3; i.e., much more phenomenal improvement is observed with the shift from $GF(4)$ to $GF(8)$ than that of the shift from $GF(8)$ to $GF(16)$.

Discussion: Approximations in deriving the asymptotic gain

According to (A.80), the asymptotic gain of the $(4/8, 8)$ RCDO code over the $(4/8, 4)$ code is calculated to be 2.26 dB whilst the $(4/8, 16)$ code enjoys another 1.60 dB gain over the $(4/8, 8)$ code, which far outvalue the largest coding gain observed in Fig. A.3. Several aspects in the calculation of the asymptotic gain above contributes to this discrepancy.

- The asymptotic gain given in (A.80) is independent of E_b/N_0. Typically, the BER is a concave function of E_b/N_0, which means that the magnitude of the slope for a BER curve increases as E_b/N_0 gets large. The asymptotic gain corresponds to the coding gain when E_b/N_0 is large; i.e., where the slopes of the BER curves are similar for different codes. However, due to limited simulation time, we were unable to record the BER performances below 10^{-7}. As is observed in Fig. A.3, the three BER curves are crossed at the E_b/N_0 of around 10^{-4}. In the E_b/N_0 region under concern, the BER curves for the three codes still exhibit large differences in their slopes; therefore, the coding gain may further increase as E_b/N_0 gets larger. Actually, from the tendency of the curves in Fig. A.3, we may expect larger coding gains if we were able to simulate to get lower BERs.
- The bit error probability given in (A.79) is based on the maximum likelihood de-

coding algorithm; i.e., suppose the codeword with weight d_{free} spans T time slots, the ML decoding calculates the likelihood of cT symbols as a whole. However, the BP decoding process applied in our RCDO codes considers the likelihood of only d_c symbols (a constraint node) each time; in addition, it uses iterative decoding, i.e., these likelihoods are updated during each round of iterations.

– In (Lin and Costello, 2004), the error probabilities of a simple convolutional code is also calculated using their *distance spectrum*, which evaluates all possible weights of the encoded sequences for a convolutional code as well as the number of the code sequences corresponding to each of these weights. The estimation of the error probabilities using the distance spectrum turned out to be close to its actual performances. However, in our derivation, due to the large constraint length of the codes, we were unable to calculate the distance spectrum of the RCDO codes; leading to several differences between our calculation and that in (Lin and Costello, 2004) : i) the error probabilities for the simple codes in (Lin and Costello, 2004) are calculated using all the distances of the convolutional code rather than the free distance alone as in our case, i.e., we only consider the first term in the distance spectrum; ii) in the distance spectrum, all the distance terms are weighted by the number of codewords giving rise to these distances, however, our P_b is calculated without knowing the number of codewords with weight d_{free}; iii) the convolutional code in (Lin and Costello, 2004) applies the Viterbi algorithm (Viterbi, 1967), which closely resembles the ML decoding, while we apply the BP algorithm for our codes.

– As mentioned earlier, although it is guaranteed that there exists a code with free distances larger than the derived lower bounds, it is highly likely for an actual code to have a free distance smaller than these bounds. That is to say, using the bounds to calculate the error probabilities corresponds to the best code in terms of d_{free}. On the other hand, our RCDO codes focus on the doubly orthogonal conditions in selection of the connection positions rather than on the distance properties.

In summary, (A.80) is not an accurate approximation for the error performances of our RCDO codes due to a number of approximations used in its derivation. However, it still provides some insight into the merits of shifting to larger alphabetical sizes; it is therefore expected that when q gets larger, the improvement on the error performances by further increasing q become less obvious.

A.4 Summary

This appendix extends various results on the distance bounds of binary linear codes onto the finite fields with order higher than 2. The principle concern is on the lower and upper bounds on the free distance of the convolutional codes. Most of these bounds are functions of code parameters such as R, q, n_E, etc. The lower bounds on the free distances of the time invariant convolutional codes are used in the calculation of the achievable coding gain by shifting to codes with large values of q. Although the calculated coding gains do not correspond to those observed in our simulations, the derivation provides some insight into the benefits of moving to codes with larger alphabetical size.

APPENDIX B

Examples of q-ary Recursive CDO Codes

Table B.1 lists some of the RCDO codes used in our simulation in the form of the parity-check matrix $\mathbf{H}^T(D)$.

Table B.1 Examples of RCDO codes

$(b/c, q)$	$\mathbf{H}^T(D)$
$(4/8, 8)$	$\begin{bmatrix} 5D^{33} & 5D^{18} & 3D^{11} & 0 \\ 0 & 5D^{14} & 4D^2 & 4D^{22} \\ 5D^{25} & 6D^{20} & 0 & D^2 \\ 2D^{10} & 5D & 0 & 7D^{26} \\ 7 & 2D^{29} & 2D^{19} & 0 \\ 0 & 2 & 2D^{13} & 4D^{32} \\ 2D^{31} & 0 & 7 & 6D^5 \\ 5D^{15} & 0 & 6D^{17} & 7 \end{bmatrix}$
$(5/10, 8)$	$\begin{bmatrix} 0 & 0 & 7D^{33} & 5D^{29} & 5D^4 \\ 6D^{33} & 0 & 0 & 5D^{24} & D^{13} \\ 3D^{22} & 6D & 6D^8 & 0 & 0 \\ 4D^{28} & 0 & 6D^{17} & 6D^9 & 0 \\ 4D^{30} & 5D^{15} & 3D^{11} & 0 & 0 \\ 4 & 6D^6 & 0 & 0 & 2D^{21} \\ 0 & 7 & 0 & D^{25} & D^{19} \\ 0 & 5D^3 & 3 & 4D^{26} & 0 \\ 5D^2 & 0 & 0 & 7 & 4D^{12} \\ 0 & 7D^{27} & 3d^5 & 0 & 4 \end{bmatrix}$
$(6/12, 8)$	$\begin{bmatrix} 0 & 6D^{10} & 0 & 0 & 5D^{18} & 6D^{22} \\ 0 & 0 & 4D^3 & 2D^{19} & 0 & 3D^{29} \\ 0 & 3D & 0 & 4D^{22} & 2D^{16} & 0 \\ 4D^{34} & 6D^2 & 0 & 5D^6 & 0 & 0 \\ 3D^4 & 0 & 0 & 0 & 4D^{28} & 2D^{30} \\ 0 & 0 & 0 & 2D^{27} & 2D^7 & 4D^{28} \\ 7 & 0 & 6D^{17} & 2D^{15} & 0 & 0 \\ 0 & 4 & 5D^9 & 0 & 0 & 2D^{25} \\ 6D^{20} & 0 & 6 & 0 & 5D^{31} & 0 \\ 7D^{13} & 5D^{12} & 0 & 4 & 0 & 0 \\ 0 & D^{14} & 6D^{21} & 0 & 6 & 0 \\ D^8 & 0 & 6D^{32} & 0 & 0 & 5 \end{bmatrix}$
$(8/16, 4)$	$\begin{bmatrix} 0 & 0 & 2D^{20} & 3D^{33} & 0 & 3D^{24} & 0 & 0 \\ 0 & 0 & D^{35} & 3D^6 & 0 & 3D^7 & 0 & 0 \\ 0 & 0 & 0 & 2D^{29} & D^{18} & 0 & 0 & 3D^{25} \\ 2D^{40} & 3D^{11} & 0 & 0 & 3D^9 & 0 & 0 & 0 \\ 0 & 0 & 0 & D^{21} & 0 & 2D & 0 & 2D^{13} \\ 2D^{23} & 0 & 2D^{31} & 0 & 2D^{26} & 0 & 0 & 0 \\ 3D^2 & 0 & 3D^{32} & 0 & 2D^{37} & 0 & 0 & 0 \\ 0 & D^{27} & 2D^{14} & 0 & 0 & 3D^3 & 0 & 0 \\ 3 & 0 & 0 & 0 & 3D^{36} & 0 & 2D^4 & 0 \\ 0 & 3 & 0 & 3D^{19} & 0 & 0 & 3D^{38} & 0 \\ D^{16} & 0 & 2 & 0 & 0 & 2D^5 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 2D^{17} & D^{20} \\ 3D^8 & 0 & 0 & 0 & 3 & 0 & 0 & 2D^{19} \\ 0 & 3D^{30} & 0 & 0 & 0 & 3 & D^{10} & 0 \\ 0 & D^{28} & 0 & 0 & 0 & 0 & 2 & 2D^{22} \\ 0 & 2D^{15} & 0 & 0 & 0 & 0 & 3D^{12} & 3 \end{bmatrix}$