

Since the inception of information theory by Shannon in 1948, a vast amount of research has been conducted concerning error correcting codes for communication over noisy channels. *Low-Density Parity-Check* (LDPC) codes represent one important class of such channel codes that are capable of closely approaching the fundamental capacity limits for sufficiently large code lengths. This thesis especially accounts for the growing heterogeneity and the varying requirements of future (mobile) communication systems by exploring flexible LDPC codes with respect to their code rate, length, and decoding complexity.

As foundation, the traditional design process of LDPC codes is reviewed, which consists of two disciplines:

- asymptotic analysis that predicts the *average performance* of a *code ensemble* and
- code construction that comprises *finite length effects* and actual *error correction capabilities*.

The following main results introduce a novel flexible design of *multi-rate* and *multi-length* LDPC codes:

- multi-rate capabilities by joint consideration of *information shortening* and *parity puncturing*,
- multi-length LDPC codes by *concurrent shortening and puncturing*,
- multi-length *quasi-cyclic* (QC) LDPC codes by *jointly optimized lifting matrix construction*.

All proposed designs are based on a single *mother code*, enabling efficient hardware implementations that reduce the complexity compared to current state-of-the-art methods. Furthermore, the results are combined into a multi-rate-and-multi-length LDPC code design that surpasses the flexibility of previously known approaches.

Advances in terms of flexible decoding algorithms and their implementations are also investigated to meet the demands of the growing variety of heterogeneous devices with different computational capabilities and their manifold requirements in terms of error correction performance, computational complexity, and convergence speed.

Flexible Low-Density Parity-Check Codes: Rate, Length, and Complexity

Moritz Beermann

Moritz Beermann · Flexible Low-Density Parity-Check Codes: Rate, Length, and Complexity

Flexible Low-Density Parity-Check Codes: Rate, Length, and Complexity

Von der Fakultät für Elektrotechnik und Informationstechnik
der Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines Doktors
der Ingenieurwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Ingenieur

Moritz Beermann

aus Münster

Berichter: Universitätsprofessor Dr.-Ing. Peter Vary
 Professor Dr.-Ing. Stephan ten Brink

Tag der mündlichen Prüfung: 20. November 2015

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.

AACHENER BEITRÄGE ZU DIGITALEN NACHRICHTENSYSTEMEN

Herausgeber:

Prof. Dr.-Ing. Peter Vary
Institut für Nachrichtengeräte und Datenverarbeitung
Rheinisch-Westfälische Technische Hochschule Aachen
Muffeter Weg 3a
52074 Aachen
Tel.: 02 41-80 26 956
Fax: 02 41-80 22 186

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar

1. Auflage Aachen:
Wissenschaftsverlag Mainz in Aachen
(Aachener Beiträge zu digitalen Nachrichtensystemen, Band 42)
ISSN 1437-6768
ISBN 978-3-95886-083-4

© 2016 Moritz Beermann

Wissenschaftsverlag Mainz
Süsterfeldstr. 83, 52072 Aachen
Tel.: 02 41 / 87 34 34
Fax: 02 41 / 87 55 77
www.Verlag-Mainz.de

Gedruckt auf chlorfrei gebleichtem Papier

D 82 (Diss. RWTH Aachen University, 2015)

Acknowledgments

This thesis was written during my time as research assistant at the *Institute of Communication Systems and Data Processing* (IND) at the *Rheinisch-Westfälische Technische Hochschule Aachen* (RWTH Aachen University).

At first, I would like to express my thanks to Prof. Dr.-Ing. Peter Vary for his continuous support and advice since my time as *Diplom* student. His many inspiring ideas and suggestions strongly contributed to the success of my work. Furthermore, his commitment as director of the institute always made the IND a motivating and pleasant place to work at.

I would further like to especially thank my co-supervisor Prof. Dr.-Ing. Stephan ten Brink for showing interest in my work and for being a great host during my visits in Stuttgart. The many discussions we had at several conferences around the world have always been fruitful and highly appreciated.

Moreover, I am grateful to all colleagues for making the IND such a uniquely enjoyable working environment. For their many contributing ideas and suggestions for improvement of my work I would like to thank Christiane Antweiler, Tobias Breddermann, Benedikt Eschbach, Matthias Rüngeler, Magnus Schäfer, Laurent Schmalen, Tim Schmitz, Birgit Schotsch, and Florian Wickert.

I would further like to thank all the students who contributed in many ways during my time at IND. Of course, I am also thankful to all non-scientific staff for always being obliging and pleasant to work with.

Finally, I express my deepest thanks to my parents and my brother for their backup, encouragement, and understanding over all the years. Our current everyday lives probably couldn't be more different but I'm still very proud of what we as a family have established throughout the years both on personal as well as professional level.

Dear Kamilla, I thank you for your love and your patience. You have stood by my side during all of my thesis work even in times that haven't been easy for yourself and for us. I really appreciate everything you have contributed to our life together.

Abstract

Since the inception of information theory by Shannon in 1948, a vast amount of research has been conducted concerning error correcting codes for communication over noisy channels. *Low-Density Parity-Check* (LDPC) codes represent one important class of such channel codes that are capable of closely approaching the fundamental capacity limits for sufficiently large code lengths. One observable trend in the conventional design process is that the closer a code performs within the given theoretical limits, the more specifically it needs to be tailored towards the exact considered transmission setup. This process, however, does not account for the growing heterogeneity and varying requirements of future (mobile) communication systems.

Facilitating this heterogeneity, an alternative design strategy is developed in this thesis that is not restricted to a single transmission scenario but instead enables a unified design of highly flexible LDPC codes. As foundation, the traditional design process of LDPC codes is reviewed, which consists of two disciplines: the asymptotic analysis that predicts the average performance of a *code ensemble* and the actual construction of a specific code comprising *finite length effects* that ultimately determine the code's error correction capabilities.

The main part of this thesis introduces and analyzes novel concepts for the design of *multi-rate* LDPC codes and *multi-length* LDPC codes, denoting codes that are based on a single *mother code* but are capable of achieving several code rates or lengths, respectively. This design enables efficient transmission and reconstruction of data at several channel qualities, delay requirements, and data rates, while reducing the complexity compared to current state-of-the-art concepts to a large extent. To achieve multi-rate capabilities, a novel joint consideration of the well-known *information shortening* and *parity puncturing* techniques is introduced, including analysis and optimization of both asymptotic ensemble performance as well as finite length effects. Furthermore, two new concepts are introduced and evaluated to add multi-length features into LDPC code design to further enhance flexibility: multi-length LDPC codes by *concurrent shortening and puncturing* and multi-length *quasi-cyclic* (QC) LDPC codes by *jointly optimized lifting matrix construction*. Both of these concepts are shown to be directly compatible with the proposed multi-rate design, resulting in combined multi-rate-and-multi-length LDPC codes that surpass the flexibility of all previously known LDPC code designs.

Advances in terms of flexible decoding algorithms and their implementations are presented to also meet the demands of the growing variety of heterogeneous devices with different computational capabilities and their manifold requirements in terms of error correction performance, computational complexity, and convergence speed. Two novel *decoder schedules* are introduced that are motivated by these kinds of flexibility requirements. Finally, the first fully generic *non-binary* LDPC decoder implementation on a *graphics processing unit* (GPU) is presented, that is capable of achieving high speedups without any algorithmic approximations.

Contents

Notations, Symbols & Abbreviations	xi
1 Introduction	1
2 Low-Density Parity-Check (LDPC) Codes	5
2.1 Low-Density Parity-Check Codes	5
2.2 Basic Definitions	7
2.3 Tanner Graph Description	11
2.4 Encoding	16
2.5 Message-Passing Decoding	19
2.6 Performance Measures	22
2.6.1 Ensemble Performance	22
2.6.2 Code Performance	23
2.7 Ensemble Optimization	25
2.7.1 Density Evolution (DE) for Binary LDPC Codes	25
2.7.2 Optimization Problem	32
2.8 Code Construction	38
2.8.1 Progressive Edge-Growth (PEG)	39
2.8.2 Improved Progressive Edge-Growth (iPEG)	39
2.9 Quasi-Cyclic (QC) LDPC Codes	40
2.10 Non-Binary LDPC Codes	43
2.10.1 Transmission System and Non-Binary LLR Vectors	43
2.10.2 Non-Binary Belief Propagation Decoding	45
2.10.3 Ensemble Optimization and Code Construction	48

3	Multi-Rate LDPC Codes	49
3.1	Information Shortening	50
3.1.1	Encoding and Decoding	50
3.1.2	Asymptotic Analysis and Optimization	53
3.1.3	Finite Length Optimization: Maximum Distance	57
3.1.4	Finite Length Optimization: Breaking Cycles	60
3.2	Parity Puncturing	67
3.2.1	Encoding and Decoding	67
3.2.2	Asymptotic Analysis and Optimization	68
3.2.3	Finite Length Optimization: Guaranteed Recoverability	73
3.3	Jointly Optimized Information Shortening and Parity Puncturing	79
3.3.1	Asymptotic Analysis and Optimization	80
3.3.2	Optimized Code Construction	89
3.3.3	Simulation Results	96
3.4	Concluding Remarks	98
3.4.1	Comparison to other Multi-Rate Schemes	100
3.4.2	Special Consideration of Non-Binary LDPC Codes	102
4	Multi-Length LDPC Codes	103
4.1	Length Reduction by Concurrent Shortening and Puncturing	105
4.1.1	Encoding and Decoding	105
4.1.2	Asymptotic Analysis and Optimization	106
4.1.3	Simulation Examples	110
4.1.4	Joint Multi-Length and Multi-Rate Considerations	115
4.1.5	Concluding Remarks	117
4.2	Multi-Length Quasi-Cyclic LDPC Codes	117
4.2.1	Encoding and Decoding	117
4.2.2	Jointly Optimized Lifting Matrix Construction	119
4.2.3	Simulation Examples	123
4.2.4	Multi-Rate and Multi-Length Quasi-Cyclic LDPC Codes	127
4.2.5	Concluding Remarks	130

5	Optimized Message-Passing Schedules	133
5.1	Probabilistic Sequential Decoding	134
5.1.1	Probabilistic Shuffled Scheduling	134
5.1.2	Probabilistic Check-Shuffled Scheduling	136
5.1.3	Adaptation to Decoding of Non-Binary LDPC Codes	136
5.1.4	Simulation Results	137
5.2	Improved Residual Belief Propagation	140
5.2.1	Residual Belief Propagation (RBP)	141
5.2.2	Novel Queue Residual Decoding (QRD)	143
5.2.3	Simulation Results	145
5.3	Concluding Remarks	149
6	GPU Implementation of Non-Binary LDPC Decoding	151
6.1	Graphics Processing Unit (GPU) Programming	151
6.2	Log-FFT Belief Propagation for Non-Binary LDPC Codes	152
6.2.1	Description of the Memory Arrangement	152
6.2.2	Description of the CUDA Kernels	153
6.3	Layered Decoding for Non-Binary QC LDPC Codes	157
6.3.1	Log-FFT Belief Propagation with Layered Schedule	158
6.3.2	Description of the Memory Arrangement	159
6.3.3	Description of the CUDA Kernels	159
6.4	Performance Evaluation	161
6.4.1	Flooding Schedule	161
6.4.2	Layered Schedule	164
7	Summary	167
A	Derivation of Maximum Average Relative Code Length	171
	Bibliography	175

Notations, Symbols & Abbreviations

Notations

Vectors, Matrices, and Scalars

$\mathbf{x}$	bold, italic lower case letters represent vectors
x_n	scalar vector elements are denoted by italic lower case letters
$\mathbf{L}$	bold, italic upper case letters represent vectors
$\mathbf{H}$	bold upper case letters represent matrices
$\mathbf{H}_{*,n}$	bold, italic upper case letters represent column vectors of a matrix
$\mathbf{H}_{m,*}$	bold, italic upper case letters represent row vectors of a matrix
$H_{m,n}$	italic upper case letters represent scalar matrix or vector elements
$(\mathbf{x}, \mathbf{p})$	concatenation of two vectors
$(\mathbf{A}, \mathbf{B})$	concatenation of two matrices
$\mathbf{H}_{[i \rightarrow j]}$	the submatrix consisting of columns i to j of $\mathbf{H}$

Sets and Intervals

$\mathcal{V}$	calligraphic upper case letters represent sets
$\{x, \dots, y\}$	discrete set of all values between x and y
$[x, y]$	continuous interval of all values between x and y including x and y
(x, y)	continuous interval of all values between x and y excluding x and y

Probabilities

$P(x = 0)$	discrete probability of $x = 0$
$p(x)$	continuous probability density function
$L(y)$	log-likelihood ratio of y
$E\{X\}$	expected value of random variable X

Miscellaneous Mathematical Functions and Operators

$\ker$	kernel of a matrix
rank	rank of a matrix
$\min$	minimum of a set
$\max$	maximum of a set
$\arg \min$	the argument at which a minimum is achieved
$\arg \max$	the argument at which a maximum is achieved
$ \mathbf{x} $	number of non-zero elements of a vector
$ \mathbf{H} $	number of non-zero elements of a matrix
$ \mathcal{V} $	cardinality of a set
$\ x\ $	the magnitude of a scalar
$\ \mathbf{x}\ $	the Euclidean norm of a vector
$\tanh$	hyperbolic tangent
atanh	inverse hyperbolic tangent
e^x	exponential function of x
$\log$	logarithm to the base e
$\log_2$	logarithm to the base 2
$\log_{10}$	logarithm to the base 10
$\log_\alpha(\alpha^i)$	yields exponent i of a Galois field element in exponential representation
$\delta(x)$	Dirac delta function
$\phi(x)$	phi function from [CRU01]
$\phi^{-1}(x)$	inverse phi function from [CRU01]
$h(s, x)$	auxiliary function in density evolution
$a \bmod b$	modulo operation
$\boxplus$	the boxplus operator
$\boxplus$	a sum using the boxplus operation
$\setminus$	exclusion from a set, e.g., $\{1, 2, 3\} \setminus \{2\} = \{1, 3\}$
$\otimes$	convolution
$(x)^{\otimes(j-1)}$	$(j-1)$ -fold convolution of x
$\approx$	approximately equal to
$\nless$	not less than
$\rightarrow$	goes to
$\cup$	union of two sets
$\bigcup$	union of multiple sets
$\forall$	for all
$\exists$	exists
$\in$	element of
$\notin$	not an element of
$\mathcal{O}(\cdot)$	complexity class in big $\mathcal{O}$ notation

$\mathcal{N}(\mu, \sigma^2)$	Gaussian distribution with mean μ and variance σ^2
$\binom{k}{j}$	binomial coefficient
$x _{\text{dB}}$	conversion of the value x to the dB scale
$\mathcal{Q}(x)$	quantized value of x
$\lceil x \rceil$	largest integer less than or equal to x
$\lfloor x \rfloor$	smallest integer greater than or equal to x
$\lceil x \rceil$	integer value closest to x (fair rounding)
$\Gamma(\cdot)$	transform of pdf to logarithmic domain from [RSU01]
$\Gamma^{-1}(\cdot)$	inverse transform of pdf from [RSU01]
$\mathcal{F}(\cdot)$	Fast Fourier Transform
$\mathcal{F}^{-1}(\cdot)$	inverse Fast Fourier Transform
$\tilde{\mathbf{x}}$	a permuted version of vector $\mathbf{x}$
$\hat{\mathbf{x}}$	an estimated version of vector $\mathbf{x}$
$\mathbf{P}^\top$	the transpose of a matrix
$\mathbf{P}^{-1}$	the inverse of a matrix

List of Principal Symbols

Latin Symbols

$\mathbf{A}$	lifting matrix of a quasi-cyclic LDPC code
$\tilde{\mathbf{A}}$	permuted lifting matrix $\mathbf{A}$
ACE_m	minimum ACE of all shortest cycles passing through the current variable node and check node m during PEG processing, assuming an edge was placed between them
$\text{ACE}_m^{[Z_i]}(n)$	minimum ACE of all shortest cycles passing through the current variable node n and check node m during PEG processing, assuming an edge was placed between them in the check matrix expanded by lifting factor Z_i
b	number of quantization bits
$\mathbf{b}_i$	basis vector of a code
c	convergence parameter of Simulated Annealing
c	counter for punctured nodes during PEG processing
$c_{m,n}$	check-to-variable message
$c_{m,n}^{(i)}$	check-to-variable message in i -th iteration
$c_{m,n}^{\text{current}}$	current value of a check-to-variable message
$c_{m,n}^{\text{updated}}$	value of a check-to-variable message after a check node update
$C(\cdot)$	cost function of optimization problem
$\mathcal{C}$	set of check nodes in a Tanner graph
$\mathfrak{C}$	a set of code words, i.e., a code
d_c	maximum check node degree (= row weight of the check matrix)
$\bar{d}_c$	average check node degree (= row weight of the check matrix)

$\mathbf{d}_c$	sequence of check node degrees (= row weights)
$d_{c,m}$	degree of check node m (= weight of row m of the check matrix)
d_v	maximum variable node degree (= column weight of the check matrix)
$\bar{d}_v$	average variable node degree (= column weight of the check matrix)
$\mathbf{d}_v$	sequence of variable node degrees (= column weights)
$d_{v,n}$	degree of variable node n (= weight of column n of the check matrix)
$\mathbf{d}_v^{\text{RC}}$	rate-compatible degree sequence
$d_{v,n}^{\text{RC}}$	degree of n variable node in rate-compatible degree sequence
$\mathbf{D}$	matrix containing pairwise distance of all variable nodes
$D_{i,j}$	distance between variable nodes i and j
$e^{(k)}$	probability of a variable-to-check message to be zero due to puncturing
E	number of edges in a Tanner graph
$E^{[rs]}$	number of edges in a shortened Tanner graph
$\mathcal{E}$	set of edges in a Tanner graph
E_b	energy spent per information symbol by the transmitter
E_s	energy spent per modulation symbol by the transmitter
$f_{c \rightarrow v}(\cdot)$	function that computes check-to-variable messages
$f_{v \rightarrow c}(\cdot)$	function that computes variable-to-check messages
$\mathbb{F}_2$	the finite field with 2 elements, i.e., the binary values $\{0, 1\}$
$\mathbb{F}_q$	the finite field (also Galois field) with q elements
$\mathbb{F}_q^+$	the finite field $\mathbb{F}_q$ with the zero element excluded
$\mathbf{g}$	gap of approximately triangulated check matrix
g	the girth, i.e., the shortest cycle in a Tanner graph
$\mathbf{g}$	vector with local girths of all variable nodes
$g_{\mathcal{I}}$	minimum cycle occurring within information nodes
$g_{\max}$	maximum local girth of all variable nodes
g_n	local girth of variable node n
$\mathbf{G}$	generator matrix
$\mathbf{G}_{[i \rightarrow j]}$	submatrix consisting of columns i to j of the generator matrix
$\mathcal{G}$	Tanner graph
$\mathbf{H}$	(parity) check matrix
$\tilde{\mathbf{H}}$	a permuted check matrix
$\mathbf{H}_{*,n}^{\top}$	a single column of the check matrix
$\mathbf{H}_{m,*}$	a single row of the check matrix
$\mathbf{H}_{[i \rightarrow j]}$	submatrix consisting of columns i to j of the check matrix
$\mathbf{H}^{[Z]}$	check matrix of a quasi-cyclic LDPC code
i^*	optimal index in Maximum Distance optimization
$\mathcal{I}$	set of information nodes in a Tanner graph
I_{avg}	average number of executed decoding iterations
I_{eff}	equivalent number of decoding iterations
$I_{\max}$	maximum number of decoding iterations
$\mathbf{I}_K$	identity matrix of dimension $K \times K$
$\mathbf{I}_s$	identity matrix cyclically shifted to the right by one position

$\mathbf{I}_s^{-1}$	square all zero matrix
$\mathbf{I}_s^{A_{m',n'}}$	identity matrix cyclically shifted to the right by $A_{m',n'}$ position
$\bar{k}$	average recoverability level of punctured nodes
k_s	relative shortened information word length
K	number of information bits per binary code word
K_d	number of shortened bits
K_s	shortened information word length
K_q	number of information symbols per non-binary code word
$\mathcal{K}_{ACE}$	set of check node candidates with maximum ACE value during PEG processing
$\mathcal{K}_{\text{girth}}$	set of check node candidates that maximize girth during PEG processing
$\mathcal{K}_{\text{weight}}$	set of check node candidates of minimum degree during PEG processing
$\mathcal{K}^{[Z_i]}$	set of all check node candidates for lifting factor Z_i during PEG processing
$\mathcal{K}_{\text{girth}}^{[Z_i]}$	set of check node candidates that maximize girth for lifting factor Z_i during PEG processing
$\mathcal{K}_{\text{weight}}^{[Z_i]}$	set of check node candidates of minimum degree for lifting factor Z_i during PEG processing
l	length of path or cycle in a Tanner graph
$\mathbf{L}$	vector containing conditional log-likelihood ratios of channel output
L_n	n -th element of vector with channel output log-likelihood ratios
$\mathbf{L}_n$	q -dimensional conditional LLR vector of n -th non-binary channel output
$L_n^{[a]}$	element of $\mathbf{L}_n$ corresponding to Galois field element a
$\mathbf{L}(\mathbf{y} \mathbf{z})$	vector containing conditional log-likelihood ratios of channel output
$\mathbf{L}(y_n z_n)$	conditional log-likelihood ratios of one channel output value
$\mathbf{L}(\tilde{\mathbf{y}} \mathbf{z})$	vector containing conditional log-likelihood ratios of channel output including values for dummy bits
$\mathbf{L}(\mathbf{y}^{\text{bin}} \mathbf{z})$	vector containing conditional log-likelihood ratios of channel output
$\mathbf{L}^q(\mathbf{y} \mathbf{z})$	vector containing q -dimensional conditional LLR vectors of channel output
$\mathbf{L}^q(y_n z_{(n-1)p+1}, \dots, z_{np})$	q -dimensional conditional LLR vector of channel output
m_p	relative punctured parity word length
M	number of parity bits per binary code word
M'	height of the lifting matrix of a quasi-cyclic LDPC code
M_d	number of punctured bits
$M_{d,i}$	number of punctured variable nodes of degree i
M_e	number of extended bits
M_p	punctured parity word length
M_q	number of parity symbols per non-binary code word
M'_q	height of non-binary quasi-cyclic lifting matrix
$\mathcal{M}(n)$	set of check nodes directly connected to variable node n
$\mathcal{M}^l(m)$	set of check nodes reachable from node m by a path of exact length l
$\mathbf{n}$	vector containing channel noise samples

n^*	optimal index in Breaking Cycles optimization
n_{eff}	effective relative code length after shortening and/or puncturing
$\bar{n}_{\text{eff}}$	average effective relative code length of a multi-rate code
$\bar{n}_{\text{eff}}^*$	maximum average effective relative code length
n_ℓ	relative reduced code length by concurrent shortening and puncturing
n_p	relative punctured code word length
n_s	relative shortened code length
N	length of a binary code word
N'	width of the lifting matrix of a quasi-cyclic LDPC code
$\bar{N}$	average code length
N_0	power spectral density of the AWGN channel
N_e	code length after extension
N_i	number of variable nodes of degree i
N_ℓ	reduced code length by concurrent shortening and puncturing
N_p	punctured code word length
N_s	shortened code word length
$N_{s,i}$	number of variable nodes of degree i after shortening
N_q	length of a non-binary code word
N'_q	width of non-binary quasi-cyclic lifting matrix
$\mathcal{N}(m)$	set of variable nodes directly connected to check node m
$\mathcal{N}^l(n)$	set of variable nodes reachable from node n by a path of exact length l
$\mathbb{N}$	the natural numbers including zero
$\mathbb{N}^+$	the natural numbers excluding zero
p	the number of bits per Galois field symbol, $q = 2^p$
$\mathbf{p}$	vector containing one parity word
$\tilde{\mathbf{p}}$	a permuted parity word
$\mathbf{p}$	boolean puncturing vector containing ones for punctured positions and zeros otherwise
$\mathbf{p}_i$	i -th entry in boolean puncturing vector $\mathbf{p}$
$\tilde{\mathbf{p}}$	permuted boolean puncturing vector
$\tilde{\mathbf{p}}_i$	i -th entry in permuted boolean puncturing vector $\tilde{\mathbf{p}}$
$\mathbf{P}$	permutation matrix
$\mathbf{P}_a$	permutation of LLR vector corresponding multiplication with Galois field element a
$\mathbf{P}_a^{-1}$	inverse permutation of LLR vector
$\mathbf{P}_c$	a column permutation matrix
$\mathbf{P}_r$	a row permutation matrix
$\mathbf{P}_{\text{RC}}$	rate-compatible permutation matrix
$\mathcal{P}$	set of parity nodes in a Tanner graph
$\mathbf{P}_{\text{error}}^{[k]}$	decoding error probability in k -th iteration
$\mathbf{P}_{\text{c,error}}^{[k]}$	probability of check-to-variable message to be in error in k -th iteration
$\mathbf{P}_n^{[g]}$	updating probability of variable node n in probabilistic decoder schedules

P_{removed}	probability of an edge being removed by shortening
$P_{\text{AWGN,LLR}}$	pmf of quantized channel log-likelihood ratios
$P_c^{[k]}$	pmf of quantized check-to-variable messages in k -th iteration
$P_{c,j}^{[k]}$	pmf of quantized outgoing messages of degree- j check nodes in k -th iteration
$P_{c,0}^{[k]}(x)$	joint pmf of check-to-variable messages that are zero in k -th iteration due to puncturing
$P_{c,\neq 0}^{[k]}(x)$	joint pmf of check-to-variable messages that are non-zero in k -th iteration
$P_{c \neq 0}^{[k]}(x)$	conditional pmf of check-to-variable messages that are non-zero in k -th iteration
$P_v^{[k]}$	pmf of quantized variable-to-check messages in k -th iteration
$P_{v,0}^{[k]}(x)$	pmf of variable-to-check messages that are zero in k -th iteration due to puncturing
$P_{v,\neq 0}^{[k]}(x)$	joint pmf of variable-to-check messages that are non-zero in k -th iteration
$P_{v \neq 0}^{[k]}(x)$	conditional pmf of variable-to-check messages that are non-zero in k -th iteration
$p_{\text{AWGN,LLR}}$	pdf of channel log-likelihood ratios
$p_c^{[k]}$	pdf of check-to-variable messages in k -th iteration
$p_{c,j}^{[k]}$	pdf of outgoing messages of degree- j check nodes in k -th iteration
$p_v^{[k]}$	pdf of variable-to-check messages in k -th iteration
$p_{v,i}^{[k]}$	pdf of outgoing messages of degree- i variable nodes in k -th iteration
q	the number of Galois field elements, $q = 2^p$
$q(r)$	minimum channel quality at which error-free decoding is possible with code rate r , also referred to channel capacity limit
$\mathcal{Q}$	message queue for (L)QRD schedules
$\mathbf{Q}_n$	non-binary <i>a posteriori</i> LLR vector
$Q_n^{[a]}$	element of $\mathbf{Q}_{m,n}$ corresponding to Galois field element a
$\mathbf{Q}_{n,m}$	non-binary variable-to-check message containing conditional LLR vector
$\tilde{\mathbf{Q}}_{n,m}$	permuted non-binary variable-to-check message containing conditional LLR vector
r	code rate
$r^\star$	mother code rate at which maximum average relative code length $\bar{n}_{\text{eff}}^\star$ is achieved
r_e	code rate after extension
r_k	k -th code rate of a multi-rate code
$r_{\min}$	minimum code rate of multi-rate code
$r_{\max}$	maximum code rate of multi-rate code
r_μ	mother code rate
r_p	punctured code rate
$r_{p,\ell}$	virtual punctured code rate of concurrent shortening and puncturing
r_s	shortened code rate

$r_{s,\ell}$	virtual shortened code rate of concurrent shortening and puncturing
r_{target}	target code rate
$\mathbf{r}(\cdot)$	residual of a message
$\mathbf{r}_{\text{max}}$	maximum residual of all messages
$\mathbf{r}_{\text{thresh}}$	adaptive threshold for residuals to be added to message queue
$R(x)$	check node degree distribution in node perspective
R_i	coefficient of check node degree distribution in node perspective
$R'(x)$	derivative with respect to x of $R(x)$
$R^{[rs]}(x)$	shortened check node degree distribution in node perspective
$R_j^{[rs]}$	coefficient of shortened check node degree distribution in node perspective
$\mathbf{R}_{m,n}$	non-binary check-to-variable message containing conditional LLR vector
$\tilde{\mathbf{R}}_{m,n}$	permuted non-binary check-to-variable message
s	auxiliary variable in Gaussian approximated density evolution
$\mathbf{s}$	ordered vector with dummy bit positions
$s^{[Z]}(m)$	shift value in lifting matrix with lifting factor Z corresponding to connection of first variable node within a group to check node m in the check matrix
$t^{[k]}$	auxiliary variable in Gaussian approximated density evolution
T	temperature (convergence parameter) of Simulated Annealing process
T_s	modulation symbol duration
$\mathcal{T}_n^l$	set of nodes reachable from node n by a path up to length l
$\mathbf{T}_{m,n}$	auxiliary message vectors in layered non-binary BP decoding
$\tilde{\mathbf{T}}_{m,n}$	permuted auxiliary message vectors in layered non-binary BP decoding
U	total number of message updates
U_{avg}	average number of executed message updates
$v_{n,m}$	variable-to-check message
$v_{n,m}^{(i)}$	variable-to-check message in i -th iteration
$\mathcal{V}$	set of variable nodes in a Tanner graph
$\mathcal{V}_{\text{active}}^{(i)}$	set of active variable nodes within a decoding iteration
$\mathcal{V}_d$	set of variable nodes that are shortened from a Tanner graph
w_k	weight representing importance of the k -th code rate of a multi-rate code
$\mathbf{x}$	vector containing one information word
$\hat{\mathbf{x}}$	vector containing estimated information word
$\tilde{\mathbf{x}}$	a permuted information word or an information word containing dummy bits
$\hat{\tilde{\mathbf{x}}}$	vector containing estimated information word including dummy bits
$\mathbf{x}^{\text{bin}}$	vector containing one binary information word
$\hat{\mathbf{x}}^{\text{bin}}$	vector containing estimated binary information word
Δx	width of quantization intervals
$\mathbf{y}$	vector containing one code word
$\hat{\mathbf{y}}$	vector containing estimated code word
$\tilde{\mathbf{y}}$	a permuted code word or a code word containing dummy bits

$\mathbf{y}^{\text{bin}}$	vector containing one binary code word
$\hat{\mathbf{y}}^{\text{bin}}$	vector containing estimated binary code word
$\mathbf{z}$	vector containing noisy channel output samples
Z	lifting factor of a quasi-cyclic LDPC code
$Z_{\min}$	minimum lifting factor of a multi-length quasi-cyclic LDPC code
$Z_{\max}$	maximum lifting factor of a multi-length quasi-cyclic LDPC code

Greek Symbols

α	the primitive element of a finite field
α	adaptive factor controlling the threshold $\mathbf{r}_{\text{thresh}}$ for residuals to be added to message queue
$\alpha_l^{\mathcal{I}}$	number of all cycles of length l incident to information nodes (counted node-wise)
α	cycle histogram of a Tanner graph
α_n	local cycle histogram of variable node n
$\alpha_{n,l}$	number of cycles of length l running through variable node n
$\beta_{n,m}$	normalization term used in non-binary belief propagation
γ	gap of ensemble threshold to the capacity limit
γ_k	gap of ensemble threshold to capacity limit at k -th code rate of a multi-rate code
γ_{ded}	gap to capacity limit of a dedicated code
γ_{BER}	gap of channel quality at which target BER is reached to capacity limit
$\gamma_{\text{BER}}^{[10^{-4}]}$	gap to capacity limit at BER 10^{-4}
$\gamma_{\text{FER}}^{[10^{-2}]}$	gap to capacity limit at FER 10^{-2}
$\gamma_{\mu,k}^{[10^{-4}]}$	gap of $\tau_{\mu,k}^{[10^{-4}]}$ to capacity limit
$\bar{\gamma}$	mean gap of ensemble thresholds to capacity limit
$\bar{\gamma}_{\text{ml}}$	mean gap to capacity limit of a multi-length code
$\bar{\gamma}_{\text{mr}}$	mean gap to capacity limit of a multi-length-by-multi-rate code
$\bar{\gamma}_{\mu}$	mean gap to capacity limit for mother code rate r_{μ}
$\bar{\gamma}_{\text{ded}}^{[10^{-4}]}$	mean gap to capacity limit at BER 10^{-4} of a set of dedicated codes
$\bar{\gamma}_{\mu}^{[10^{-4}]}$	mean gap to capacity limit at BER 10^{-4} of multi-rate code with mother code rate r_{μ}
$\Delta\bar{\gamma}^{[10^{-4}]}$	difference between mean gaps in dB of two multi-rate setups
$\Delta^{[rs]}(x)$	degree distribution of shortened nodes in node perspective
$\Delta_i^{[rs]}$	coefficient of degree distribution of shortened nodes in node perspective
$\varepsilon^{(k)}$	probability of a check-to-variable message to be zero due to puncturing
ζ	number of supported code rates of a multi-rate code
η	ACE spectrum
η_n	local ACE spectrum of variable node n
$\tilde{\theta}_{m,n}$	permuted non-binary check-to-variable message in signed log domain
$\tilde{\theta}_{m,n}^s$	sign of $\tilde{\theta}_{m,n}$

$\tilde{\theta}_{m,n}^m$	magnitude of $\tilde{\theta}_{m,n}$
$\tilde{\Theta}_{m,n}$	Fourier transform of $\tilde{\theta}_{m,n}$
$\tilde{\Theta}_{m,n}^s$	sign of $\tilde{\Theta}_{m,n}$
$\tilde{\Theta}_{m,n}^m$	magnitude of $\tilde{\Theta}_{m,n}$
$\lambda(x)$	variable node degree distribution in edge perspective
λ_i	coefficient of variable node degree distribution in edge perspective
$\lambda'(x)$	derivative with respect to x of $\lambda(x)$
$\tilde{\lambda}$	candidate solution of λ during optimization
λ^*	local optimum of λ during optimization
$\lambda^{[rs]}(x)$	shortened variable node degree distribution in edge perspective
$\lambda_i^{[rs]}$	coefficient of shortened variable node degree distribution in edge perspective
$\Lambda(x)$	variable node degree distribution in node perspective
Λ_i	coefficient of variable node degree distribution in node perspective
$\Lambda'(x)$	derivative with respect to x of $\Lambda(x)$
$\Lambda^{[rs]}(x)$	shortened variable node degree distribution in node perspective
$\Lambda_i^{[rs]}$	coefficient of shortened variable node degree distribution in node perspective
μ	index of mother code rate
μ_L	mean of channel log-likelihood ratios
$\mu_c^{[k]}$	mean of check-to-variable messages in k -th iteration
$\mu_{c,j}^{[k]}$	mean of outgoing messages of degree- j check nodes in k -th iteration
$\mu_v^{[k]}$	mean of variable-to-check messages in k -th iteration
$\mu_{v,i}^{[k]}$	mean of outgoing messages of degree- i variable nodes in k -th iteration
ξ	number of supported code lengths of a multi-length code
$\Pi^{[rp]}(x)$	puncturing polynomial at code rate r_p
$\Pi_i^{[rp]}$	coefficient of puncturing polynomial at code rate r_p
$\tilde{\Pi}^{[rp]}(x)$	normalized puncturing polynomial at code rate r_p
$\tilde{\Pi}_i^{[rp]}$	coefficient of normalized puncturing polynomial at code rate r_p
$\check{\Pi}^{[n\ell]}(x)$	scaled puncturing polynomial for concurrent shortening and puncturing
$\check{\Pi}_i^{[n\ell]}$	coefficient of scaled puncturing polynomial for concurrent shortening and puncturing
ϱ	density of entries in a matrix
$\rho(x)$	check node degree distribution in edge perspective
ρ_i	coefficient of check node degree distribution in edge perspective
$\rho'(x)$	derivative with respect to x of $\rho(x)$
$\tilde{\rho}$	candidate solution of ρ during optimization
ρ^*	local optimum of ρ during optimization
$\rho^{[rs]}(x)$	shortened check node degree distribution in edge perspective
$\rho_j^{[rs]}$	coefficient of shortened check node degree distribution in edge perspective

σ_L^2	variance of channel log-likelihood ratios
σ_n^2	variance of AWGN channel noise
τ	ensemble threshold
$\tilde{\tau}$	ensemble threshold of candidate solution during optimization
τ^*	local optimum of the ensemble threshold
$\tau^{[\text{BER}]}$	channel quality at which BER curve reaches target BER
τ_{DDE}	ensemble threshold computed by discretized density evolution
τ_{GADE}	ensemble threshold computed by Gaussian approximated density evolution
τ_k	ensemble threshold at k -th code rate of a multi-rate code
$\tau(\lambda, \rho)$	threshold of (λ, ρ) ensemble
$\tau(\Lambda, R)$	threshold of (Λ, R) ensemble
$\tau(\Lambda, R, \Pi^{[rp]})$	threshold of (Λ, R) ensemble if punctured with $\Pi^{[rp]}$
$\tau_{\mu, k}^{[10^{-4}]}$	channel quality at which multi-rate code with mother code rate r_μ reaches BER 10^{-4} for effective code rate r_k
$\Delta\tau _{\text{dB}}$	the difference in dB of a candidate threshold to the current optimum during optimization
$\tilde{\varphi}_{n,m}$	permuted non-binary variable-to-check message in signed log domain
$\tilde{\varphi}_{n,m}^s$	sign of $\tilde{\varphi}_{n,m}$
$\tilde{\varphi}_{n,m}^m$	magnitude of $\tilde{\varphi}_{n,m}$
$\tilde{\Phi}_{n,m}$	Fourier transform of $\tilde{\varphi}_{n,m}$
Φ^{exp}	expanded ACE spectrum
Φ^{cyc}	ACE-cycle histogram
$\chi_{i,l}^{(k)}$	auxiliary term used in punctured density evolution
ω_m	distance of current variable node to check node m during PEG processing

List of Abbreviations

∞ -SR	∞ -step-recoverable
ACE	approximate cycle extrinsic message degree
AMR	Adaptive Multi-Rate
ARBP	approximate residual belief propagation
ascPEG	ascending progressive edge-growth
AWGN	additive white Gaussian noise
BER	bit error rate
BIAWGN	binary input additive white Gaussian noise
BP	belief propagation
BPSK	binary phase-shift keying
BreakCyc	breaking cycles
CNP	check node processor
CPU	central processing unit
CUDA	Compute Unified Device Architecture

dB	decibel
DDE	discretized density evolution
DE	density evolution
DVB	Digital Video Broadcast
EMD	extrinsic message degree
EXIT	extrinsic information transfer
FEC	forward error correction
FER	frame error rate
FFT	Fast Fourier Transform
GA	Gaussian approximation
GADE	Gaussian approximated density evolution
GPU	graphics processing unit
GSM	Global System for Mobile Communications
HARQ	hybrid automatic repeat request
i.i.d.	independent and identically distributed
IDS	informed dynamic scheduling
IEEE	Institute of Electrical and Electronics Engineers
iPEG	improved progressive edge-growth
IRA	irregular repeat accumulate
k -SR	k -step-recoverable
LDPC	Low-Density Parity-Check
LLR	log-likelihood ratio
LTE	Long Term Evolution
LQRD	lazy queue residual decoding
MD-DBI	maximum distance dummy bit insertion
ML	maximum-likelihood
ML-by-MR	multi-length by multi-rate
ModBreakCyc	modified breaking cycles
MuLiPEG	multi-length improved progressive edge-growth
MuLQPEG	multi-length quasi-cyclic progressive edge-growth
MuRiPEG	multi-rate improved progressive edge-growth
MuRMuLQiPEG	multi-rate multi-length quasi-cyclic improved progressive edge-growth
NB	non-binary
NS	node-wise scheduling
NWRBP	node-wise residual belief propagation
pdf	probability density function
PEG	progressive edge-growth
pmf	probability mass function
QC	quasi-cyclic
QRD	queue residual decoding
R-DBI	random dummy bit insertion
RAM	random-access memory
RBP	residual belief propagation

SA	Simulated Annealing
SimpBreakCyc	simplified breaking cycles
SNR	signal to noise ratio
uBER	undetected bit error rate
uFER	undetected frame error rate
UMTS	Universal Mobile Telecommunications System
VNP	variable node processor
w.l.o.g.	without loss of generality
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network

Introduction

The main driver in the development of the 3rd and 4th generations of mobile cellular networks (3G and 4G) was – and still is – the exponentially increasing traffic volume that needs to be delivered to the rapidly increasing number of mobile users. This trend of exponential traffic growth will further continue and more densely populated smaller cells have to provide increased peak data rates. At the same time, the coverage should be improved also for users that are quickly moving or located at the cell edges. Besides the traffic growth various additional challenges are predicted to dominate the evolution of the next mobile network generation 5G [Eri13, CFH14]. One key challenge is the expanding heterogeneity in many aspects of future communication systems. Not only will there be more and more devices, but also the types of devices will diversify. While smart phones, tablets, and laptop computers become more powerful, the vision of the *Internet of Things* and a strong increase in *Machine to Machine* communication will accommodate a variety of new devices serving new applications. This evolution of the mobile networks comes with characteristics and requirements that differ from the current human-centric design. Safety-related applications, e.g., in traffic control, electrical grid management, or industrial production, might require guaranteed latencies of only a few milliseconds. Low-cost battery-powered sensor networks, in contrast, are less latency dependent but need to transmit very small amounts of data with very little energy consumption.

A central component of each communication system is its forward error correction (FEC) functionality. The inherent noise that is present on every transmission channel requires protection by *channel codes* to enable reliable communication. Since Shannon’s fundamental information theoretic results [Sha48] on the maximum achievable transmission rate for communication over noisy channels, decades of research on channel coding schemes have passed. A major breakthrough on the way to the capacity limits was marked by the advent of iterative decoding techniques in the 90s. The invention of *Turbo codes* [BGT93] and shortly thereafter the rediscovered of *Low-Density Parity-Check* (LDPC) codes [Gal62, Gal63, MN95, Mac99] have established the first practically decodable channel codes to achieve performance within fractions of a decibel (dB) of the theoretical limits for sufficiently large code lengths [tB00, CFRU01]. The iterative exchange of so called *extrinsic soft information* enables superior error correction performance, even though these iterative decoders are often not optimal

in a *maximum-likelihood* (ML) sense. The superior performance together with the linear decoding complexity (in the code length) are the key features that have led to the success of *iterative channel decoding*.

Conventional FEC code designs do not account for the described heterogeneity but focus on the optimization of codes with a fixed code rate determined by a worst-case channel condition at which the system should still be able to operate. The key drawback of this design approach is that the system is not able to benefit from channel qualities better than the one it was designed for. However, such systems typically need to be operated well above the worst-case channel condition to avoid outages due to the probabilistic variations in the channel quality over time. As a result, most of the time the channel code rate is lower than necessary, resulting in a waste of resources and a decreased throughput of user data. For a more efficient use of the available resources, it is indispensable for FEC schemes to support multiple channel code rates providing more flexibility to adapt to different channel conditions. Furthermore, it is advantageous to operate at multiple code lengths. In principle, long codes are desirable as they can achieve better error correction performance than shorter codes. At the same time, however, the length of a code implicates a certain algorithmic delay caused by the block-wise processing of code words. For applications with delay constraints – and especially for those with low transmission data rates – it can, thus, be necessary to restrict the code length to a certain maximum value.

In current LDPC code based standards like IEEE 802.16 (WiMAX [IEE09a]), IEEE 802.11 (WLAN [IEE09b]), or different *Digital Video Broadcast* (DVB) standards (DVB-T2 [Eur09b], DVB-S2 [Eur09a]), separate codes are specified for the supported code rates and lengths (with up to 22 codes in DVB-S2). While each of these codes was optimized with respect to its specific code rate and length, transmitters and receivers have to support all specified codes, which hinders an efficient implementation and leads to an increased hardware complexity.

This thesis introduces new design methods for LDPC codes that rely only on a single encoder and decoder implementation and still account for the flexibility requirements of heterogeneous transmission systems. Several original contributions towards *multi-rate* and *multi-length* LDPC codes with new dimensions of flexibility in terms of code rate and length are presented. Additionally, advances in terms of decoding algorithms that are adjustable to different requirements, e.g., low residual error rates, low complexity, or fast convergence, and their implementation are presented to account for devices with different hardware capabilities.

Structure and Contributions of the Thesis

In Chapter 2, a foundation for the subsequent contributions is laid, by introducing the considered transmission system model and reviewing the basic definitions and properties of LDPC codes. The typical two-step process of designing LDPC codes is presented. The first step consists of the optimization of the *code ensemble performance*, which is facilitated by tools like *density evolution* (DE) [RU01b] or *extrinsic*

information transfer (EXIT) *charts* [tB01]. In this context, a novel ensemble optimization technique is introduced that combines density evolution with a *Simulated Annealing* (SA) [KGJV83] approach and is particularly suitable for the multi-rate optimization presented further below. In the second step of the LDPC code design process, a specific code instance is chosen from an ensemble by constructing an actual *parity check matrix* with desirable properties, i.e., so-called *finite length effects*. At last, the basic concepts of the practically relevant cases *quasi-cyclic* (QC) LDPC codes and *non-binary* (NB) LDPC codes are covered.

Chapter 3 includes the main contributions towards a new design of multi-rate LDPC codes. These codes are based on a single so-called *mother code* but are capable of achieving several code rates, and thus enable an efficient reconstruction of data for several transmission channel qualities. They are attractive candidates to improve currently specified standards by replacing the multiple dedicated LDPC codes with a single multi-rate code. This way, more specifically optimized hardware implementations can be employed, as they can be tailored towards a single LDPC code, rather than having to encode and decode several different codes. While various known methods exist to adapt the code rate of LDPC codes, all of them are either restricted to specially structured LDPC codes or can only support good error correction performance for limited ranges of code rates.

Striving for an improved and more generic multi-rate design method, the well known *information shortening* and *parity puncturing* techniques are revisited, where information shortening decreases and parity puncturing increases the code rate of a given mother code. The contributions include a detailed asymptotic analysis and optimization of both methods. Furthermore, two novel finite length shortening schemes and a novel puncturing scheme that allows puncturing to arbitrarily high code rates are introduced. Extending the fundamental ideas of [TJ05], a novel joint ensemble optimization of shortening and puncturing leads to multi-rate LDPC codes that achieve arbitrary code rate ranges between 0 and 1. In addition to these asymptotic results, the optimized check matrix construction method *multi-rate improved progressive edge-growth* (MuRiPEG) is introduced. Together with the asymptotic optimization results, this method is shown to offer a novel degree of flexibility in terms of the supported code rate range and at the same time to outperform previously known multi-rate LDPC code construction methods.

In Chapter 4 novel design methods for multi-length LDPC codes are presented. As a subclass of *linear block codes*, LDPC codes have an inherently fixed code length that needs to be chosen at the time of code construction. Yet, the code length entails a trade-off between potentially capacity approaching performance at large code lengths and low delay features at small code lengths. To still offer flexibility at operation time, two methods for the construction of multi-length LDPC codes are illustrated that are only based on a single encoder and decoder for the mother code.

The first presented method introduces and analyzes the concept of *code length reduction by concurrent shortening and puncturing*. This approach combines two techniques that are originally intended for code rate adaptation to reduce the code length of a mother LDPC code while keeping its code rate constant. Again, asymptotic analysis

and optimization techniques are presented. The proposed check matrix construction method *multi-length improved progressive edge-growth* (MuLiPEG) is shown to be suitable to flexibly reduce the code length by up to one order of magnitude

The second method offers even more flexibility regarding the supported code lengths but is restricted to the practically relevant case of QC LDPC codes. A joint optimization of the QC LDPC code's *lifting matrix* for *expansion* to parity check matrices of several sizes is proposed by *multi-length quasi-cyclic progressive edge-growth* (MuLQPEG). The resulting multi-length QC LDPC code supports code length ranges spanning more than two orders of magnitude. At the same time, it offers error correction performance that is nearly identical to that of separately optimized codes. This approach is further combined with the multi-rate functionality, yielding multi-rate-and-multi-length QC LDPC codes with flexibility surpassing all previous approaches and exceptional error correction performance.

Chapter 5 presents advances in terms of decoding algorithms by introducing novel *decoder schedules* for *belief propagation* (BP) decoding. Due to the growing variety of devices with different computational capabilities that participate in mobile networks, decoding algorithms that are flexibly adjustable to different requirements, e.g., low residual error rates, low complexity, or fast convergence, are of special interest. The first presented *probabilistic sequential* decoder schedule combines fast convergence of iterative decoding with improved error correction performance while inducing only negligible additional complexity. The second contribution introduces *(lazy) queue residual decoding* ((L)QRD) as a generalization of recently developed *informed dynamic scheduling* (IDS) techniques [CGW10]. While these IDS strategies generally inflict a significant amount of additional complexity compared to the conventional BP decoder, they achieve much faster convergence in return. The proposed (L)QRD schemes reduce the computational overhead of previous IDS strategies to a great extent. At the same time they maintain the fast convergence properties and are even capable of achieving lower residual error rates.

Finally, in Chapter 6 a fully generic non-binary BP decoder implementation on a *graphics processing unit* (GPU) is presented. While close-to-capacity error correction performance is possible with long binary LDPC codes, especially for moderate code lengths the performance can be further improved by non-binary (NB) LDPC codes over higher order Galois fields [DM98a]. This performance gain increases with larger Galois field orders but comes at the cost of a rapidly growing decoding complexity. To speed up the decoding process, the proposed decoder architecture exploits novel ways of massive parallelization without any algorithmic approximations or requirements regarding the code structure. It is the first decoding architecture that generally applies to arbitrary NB LDPC codes and achieves large speedup values while providing exact BP performance.

Parts of the results of this thesis have been pre-published in the following references: [Bee10, BSV11, BV11, BBV11, BV13, BMSV13, BV14, BWV14, BMSV15]. These references are marked by an underlined label throughout the thesis.

Low-Density Parity-Check Codes

This chapter describes the transmission system that is considered throughout this thesis and provides an overview of *Low-Density Parity-Check* (LDPC) codes, including encoding, decoding, typical performance measures, and some code design methods.

2.1 Low-Density Parity-Check Codes

This thesis focuses on the generic design of flexible LDPC codes for the protection against transmission errors of arbitrary binary data content, transmitted over a noisy *Additive White Gaussian Noise* (AWGN) channel, also denoted BIAWGN channel in case of binary input values. Real wireless transmission channels exhibit additional detrimental effects which need to be taken care of by *channel estimation*, *equalization*, and *demodulation* which, however, are not in the focus of this thesis. Nevertheless, conducting these operations prior to channel decoding leads to an equivalent channel that can be accurately modeled as BIAWGN channel with a constant noise power per transmitted block of data. How exactly this noise power changes over time depends on the exact considered transmission channel but is not of interest in this thesis, since the design of flexible LDPC codes for different scenarios is dealt with but not any control schedules to decide which code configuration to use at any given moment.

The considered baseband transmission system model, which is depicted in Figure 2.1, takes information words $\mathbf{x} = (x_1, \dots, x_K)$ containing K independent and identically distributed (i.i.d.) bits¹ with equiprobable values 0 and 1 as input. These information words $\mathbf{x}$ are mapped to code words $\mathbf{y} = (y_1, \dots, y_N)$ of length N by an LDPC encoder (Section 2.4) of *code rate*

$$r = \frac{K}{N}. \quad (2.1)$$

¹It should be noted that throughout this thesis, the transmitted data is always in binary representation independent of what kind of LDPC code is used. In case of non-binary LDPC coding over $\mathbb{F}_q$, before encoding and decoding $p = \log_2 q$ binary values are mapped to one q -ary symbol, and likewise after encoding and decoding each q -ary symbol is mapped back to p values in binary representation (see Section 2.10).

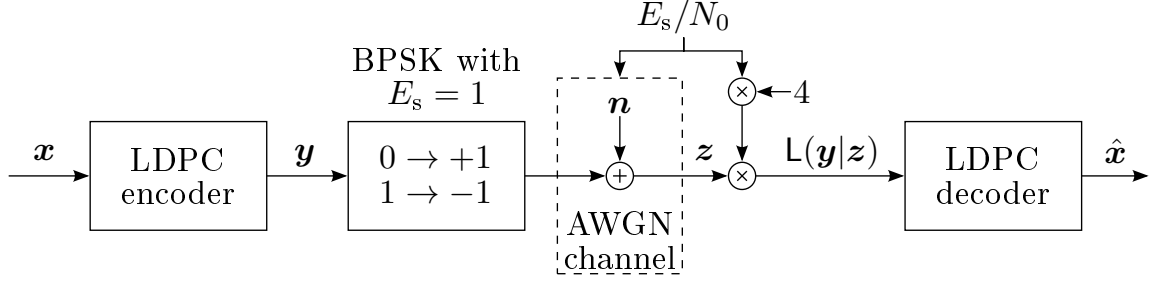


Figure 2.1: Baseband transmission system considered throughout this thesis with LDPC coding, BPSK modulation, and AWGN channel.

Each code bit is mapped to a *Binary Phase Shift Keying* (BPSK) symbol ($0 \rightarrow +1$ and $1 \rightarrow -1$). Since no cases of joint channel coding and modulation are explored in this thesis, it suffices to consider only BPSK modulation as no additional insights for the channel code can be expected by using higher order modulation constellations. The resulting signal has constant power 1, yielding the constant transmission energy per modulation symbol of $E_s = 1$ (per normalized symbol duration $T_s = 1$). The code words in this bipolar representation are subject to an additive noise vector $\mathbf{n}$ containing N random i.i.d. noise values with Gaussian distribution of zero mean and variance

$$\sigma_n^2 = \frac{N_0}{2} = \frac{1}{2E_s/N_0} \quad (2.2)$$

(as $E_s = 1$ is assumed), where $\frac{E_s}{N_0}$ denotes the *channel signal-to-noise ratio* (SNR) and is commonly used to specify the channel quality. In some cases, the alternative channel quality $\frac{E_b}{N_0}$ is used for a fair comparison, with $E_b = \frac{E_s}{r}$ denoting the average energy spent per information bit by the transmitter², whereby the effect of reducing the throughput by reducing the code rate is taken into account. At the receiver, matched filtering and inter symbol interference free sampling is assumed. Soft demodulation is then realized by multiplying the received noisy samples $\mathbf{z}$ with $4\frac{E_s}{N_0}$, yielding the vector of (conditional) channel *log-likelihood ratios* (LLRs) [HOP96]

$$\begin{aligned} \mathbf{L}(\mathbf{y}|\mathbf{z}) &= (\mathbf{L}(y_1|z_1), \dots, \mathbf{L}(y_N|z_N)) \\ &= \left(\log \frac{\mathbf{P}(y_1 = 0|z_1)}{\mathbf{P}(y_1 = 1|z_1)}, \dots, \log \frac{\mathbf{P}(y_N = 0|z_N)}{\mathbf{P}(y_N = 1|z_N)} \right) \\ &= \left(4\frac{E_s}{N_0}z_1, \dots, 4\frac{E_s}{N_0}z_N \right) =: (L_1, \dots, L_N) = \mathbf{L}, \end{aligned} \quad (2.3)$$

where equiprobable values of the code bits $\mathbf{P}(y_n = 0) = \mathbf{P}(y_n = 1) = 0.5$ are assumed. In the following, this vector $\mathbf{L}$ will be referred to as the vector of channel LLRs L_n . Finally, a soft decision LDPC decoder (Section 2.5) is fed with these channel LLRs to produce an estimate $\hat{\mathbf{x}}$ of the originally sent information word $\mathbf{x}$.

²Since only additive noise but no attenuation of the transmitted symbols is modeled here, the average energy spent per information bit by the transmitter is equal to the average energy that is available per information bit at the receiver.

The remaining sections of this chapter give an introduction to LDPC codes and their application to the considered transmission system, including some renowned state-of-the-art methods for encoding, decoding, and code design. While the described scenario is in principle well-investigated, superior performance is usually only achieved for a fixed setup, including a dedicated code designed with fixed input block length K , a fixed code rate r , and a fixed channel quality $\frac{E_s}{N_0}$. In Chapters 3 and 4 several methods are developed to overcome these restrictions by introducing *multi-rate* code design, *multi-length* code design, and combinations thereof.

2.2 Basic Definitions

For ease of notation, most definitions will be given specifically for binary LDPC codes. However, in most cases the same definitions also hold for non-binary LDPC codes if the finite field $\mathbb{F}_2 = \{0, 1\}$ is replaced by the finite field $\mathbb{F}_q = \{0, 1, \alpha, \dots, \alpha^{q-2}\}$. It will be mentioned whenever restrictions to only the binary case apply. Furthermore, all required modifications for non-binary LDPC codes will be summarized in Section 2.10.

Linear Block Codes

For $K, N \in \mathbb{N}^+$ and³ $K < N$, a binary linear (N, K) block code over the finite field $\mathbb{F}_2$ has the *code rate* $r = \frac{K}{N}$, where K is denoted the *information (word) length* and N is denoted the *code (word) length*. The code is defined by a K -dimensional linear subspace of the vector space $\mathbb{F}_2^N$. This subspace constitutes the set $\mathbf{C}$ of valid *code words* which contains 2^K distinct row vectors $\mathbf{y} = (y_1, \dots, y_N) \in \mathbb{F}_2^N$ of length N (including the all-zero vector $(0, \dots, 0)$). A corresponding encoder function realizes a bijective mapping of the set of all 2^K possible *information words* $\mathbf{x} = (x_1, \dots, x_K) \in \mathbb{F}_2^K$ of length K to the set $\mathbf{C}$ of code words. Since the code subspace is K -dimensional, this mapping can be described by means of K linearly independent vectors $\mathbf{b}_1, \dots, \mathbf{b}_K$ of $\mathbf{C}$ forming a basis of the subspace. Arranging these K basis vectors of length N as rows of a matrix results in the *generator matrix*

$$\mathbf{G} = \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_K \end{pmatrix} = \begin{pmatrix} G_{1,1} & \cdots & G_{1,N} \\ \vdots & \ddots & \vdots \\ G_{K,1} & \cdots & G_{K,N} \end{pmatrix} \in \mathbb{F}_2^{K \times N}. \quad (2.4)$$

The mapping of an information word $\mathbf{x}$ to a code word $\mathbf{y}$ is then given by

$$\mathbf{y} = \mathbf{x} \cdot \mathbf{G}, \quad (2.5)$$

³The notation $(\cdot)^+ = (\cdot) \setminus 0$ denotes the exclusion of the zero element from a set. Accordingly, $\mathbb{N}^+ = \mathbb{N} \setminus 0 = \{1, 2, 3, \dots\}$ denotes the set of all positive integers and $\mathbb{F}_q^+ = \mathbb{F}_q \setminus 0 = \{1, \alpha, \dots, \alpha^{q-2}\}$ denotes all non-zero elements of $\mathbb{F}_q$.

and the set $\mathbf{C}$ of code words $\mathbf{y}$ is given by the row space of $\mathbf{G}$:

$$\mathbf{C} = \{\mathbf{y} \in \mathbb{F}_2^N : \mathbf{y} = \mathbf{x} \cdot \mathbf{G}, \mathbf{x} \in \mathbb{F}_2^K\}. \quad (2.6)$$

In general, for one code $\mathbf{C}$, different generator matrices exist, since different bases of $\mathbf{C}$ can be found. Slightly abusing this definition, throughout this thesis, a code is referred to as a concrete, fixed mapping, and is thus described by a unique generator matrix. Most commonly this unique description is chosen as the systematic form of a generator matrix

$$\mathbf{G} = (\mathbf{I}_K, \mathbf{G}_{[K+1 \rightarrow N]}) = \underbrace{\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}}_{\mathbf{I}_K} \underbrace{\begin{pmatrix} G_{1,K+1} & \cdots & G_{1,N} \\ G_{2,K+1} & \cdots & G_{2,N} \\ \vdots & \ddots & \vdots \\ G_{K,K+1} & \cdots & G_{K,N} \end{pmatrix}}_{\mathbf{G}_{[K+1 \rightarrow N]}}, \quad (2.7)$$

where $\mathbf{I}_K$ denotes the $K \times K$ identity matrix and $\mathbf{G}_{[i \rightarrow j]}$ denotes the submatrix consisting of columns i to j of $\mathbf{G}$. In this form a code word contains the information word $\mathbf{x}$ in the first K positions and the *parity word* $\mathbf{p} = (p_1, \dots, p_M)$ with $M = N - K$ in positions $K + 1$ to N :

$$\mathbf{y} = (y_1, \dots, y_K, y_{K+1}, \dots, y_N) = (x_1, \dots, x_K, p_1, \dots, p_M) = (\mathbf{x}, \mathbf{p}). \quad (2.8)$$

The so-called *parity-check matrix* (for short *check matrix*) is denoted

$$\mathbf{H} = (\mathbf{H}_{*,1}^\top, \dots, \mathbf{H}_{*,N}^\top) = \begin{pmatrix} \mathbf{H}_{1,*} \\ \vdots \\ \mathbf{H}_{M,*} \end{pmatrix} = \begin{pmatrix} H_{1,1} & \cdots & H_{1,N} \\ \vdots & \ddots & \vdots \\ H_{M,1} & \cdots & H_{M,N} \end{pmatrix} \in \mathbb{F}_2^{M \times N}, \quad (2.9)$$

where $\mathbf{H}_{*,n}^\top = (H_{1,n}, \dots, H_{M,n})^\top$ denotes the n -th column vector of $\mathbf{H}$, while the m -th row vector of $\mathbf{H}$ is denoted $\mathbf{H}_{m,*} = (H_{m,1}, \dots, H_{m,N})$. The check matrix of a code with generator matrix $\mathbf{G}$ is defined as any full rank matrix from $\mathbb{F}_2^{M \times N}$ for which

$$\mathbf{G} \cdot \mathbf{H}^\top = \mathbf{0} \quad (2.10)$$

holds. Since for a given matrix $\mathbf{G}$ there are NM unknown entries in $\mathbf{H}$ but only KM equations in (2.10), the system is under-determined with potentially $NM - KM = M^2$ degrees of freedom, and up to $2^{(M^2)}$ different solutions may exist. For each code word

$$\mathbf{y} \cdot \mathbf{H}^\top = \mathbf{0} \quad \Leftrightarrow \quad \begin{cases} \mathbf{y} \cdot \mathbf{H}_{1,*}^\top = \sum_{n=1}^N y_n H_{1,n} = 0 \\ \vdots \\ \mathbf{y} \cdot \mathbf{H}_{M,*}^\top = \sum_{n=1}^N y_n H_{M,n} = 0 \end{cases} \quad (2.11)$$

holds, where $\mathbf{y} \cdot \mathbf{H}_{m,*}^\top = 0$ is called the m -th (*parity*-) *check equation*. The set of code words is thus given by the kernel of the check matrix

$$\mathbf{C} = \ker(\mathbf{H}) = \{\mathbf{y} \in \mathbb{F}_2^N : \mathbf{y} \cdot \mathbf{H}^\top = \mathbf{0}\}. \quad (2.12)$$

Low-Density Parity-Check Codes

A *Low-Density Parity-Check code* is a linear block code with a *sparse* check matrix $\mathbf{H}$, i.e., the number of non-zero elements is much smaller than the number of zero elements in $\mathbf{H}$, or equivalently, the *density* $\varrho = \frac{|\mathbf{H}|}{MN}$ is close to zero. The operator $|\cdot|$ denotes the number of non-zero entries if the argument is a vector or a matrix, and the cardinality (i.e., the number of elements) if the argument is a set. This low density of the check matrix is in fact the key innovation of LDPC codes, as it enables low complexity iterative decoding (cf. Section 2.5). To ensure this important sparseness property, LDPC codes are designed solely based on their check matrix. The fact that finding the sparsest check matrix for a given generator matrix is an *NP-complete* problem [Moo05] renders any generator matrix based approach practically infeasible. Even though multiple check matrices might define the same set of code words, in this thesis each check matrix is referred to as specifying a unique LDPC code. The reason for this is that iterative decoders which are based on a check matrix description potentially produce different results for different check matrices, even though these define the same code $\mathbf{C}$ according to (2.12). Furthermore, only the systematic representation of LDPC codes with code word structure according to (2.8) is considered.

As mentioned earlier, the check matrix $\mathbf{H}$ is commonly assumed to have full rank $\text{rank}(\mathbf{H}) = M$. For matrices with a rank defect, a smaller full rank matrix of dimension $\text{rank}(\mathbf{H}) \times N$ can be constructed by removing $M - \text{rank}(\mathbf{H})$ linearly dependent rows, which then defines an $(N, N - \text{rank}(\mathbf{H}))$ LDPC code of increased code rate $r = \frac{N - \text{rank}(\mathbf{H})}{N}$. Although some LDPC code construction methods explicitly use check matrices with rank defect, only full rank check matrices are considered throughout this thesis unless stated otherwise.

The sparsity of the check matrix can be exploited to efficiently store its entries using one short list of indices per row (or column, respectively). The set

$$\mathcal{N}(m) = \{n: H_{m,n} \neq 0\} \quad (2.13)$$

contains the indices of all columns that have a non-zero entry in row m , i.e., the positions of all code word symbols that participate in the m -th check equation. Likewise,

$$\mathcal{M}(n) = \{m: H_{m,n} \neq 0\} \quad (2.14)$$

contains the indices of all rows with a non-zero entry in column n , i.e., the indices of all check equations in which the n -th code word symbol participates. This way the memory requirement is reduced from $\mathcal{O}(N^2)$ memory units⁴, which are needed to store a general (dense) matrix, to only $\mathcal{O}(N)$. For *non-binary* codes, i.e., for $q > 2$, in addition to the position also the value of each non-zero element has to be stored in a similar fashion, which is, however, still possible with $\mathcal{O}(N)$ memory locations.

⁴The *big O notation* $\mathcal{O}(\cdot)$ is used throughout this thesis to classify the complexity of algorithms both in terms of runtime and memory requirements.

Regular and Irregular LDPC Code Ensembles

When originally proposed in [Gal62], only the class of *regular* (d_v, d_c) LDPC codes was considered, in which all columns of the check matrix $\mathbf{H}$ have the same (*Hamming weight* (i.e., number of non-zero entries)

$$|\mathbf{H}_{*,1}| = \cdots = |\mathbf{H}_{*,N}| = d_v, \quad (2.15)$$

and also all rows have equal weight

$$|\mathbf{H}_{1,*}| = \cdots = |\mathbf{H}_{M,*}| = d_c. \quad (2.16)$$

A regular (d_v, d_c) LDPC code (with a full rank check matrix) has the code rate

$$r = 1 - \frac{d_v}{d_c}, \quad (2.17)$$

which directly follows from the fact that the total number of non-zero entries $|\mathbf{H}|$ must be the same when counted column-wise or row-wise:

$$\begin{aligned} |\mathbf{H}| &= \sum_{n=1}^N |\mathbf{H}_{*,n}| = \sum_{m=1}^M |\mathbf{H}_{m,*}| \\ \Leftrightarrow & \quad Nd_v = Md_c \\ \Leftrightarrow & \quad 1 - \frac{d_v}{d_c} = \frac{N - M}{N} = r. \end{aligned} \quad (2.18)$$

The class of *irregular* LDPC codes was introduced much later in [LMSS98a, LMSS98b, LMSS01] and allows an arbitrary weight $|\mathbf{H}_{*,n}| = d_{v,n}$ for each column n and an arbitrary weight $|\mathbf{H}_{m,*}| = d_{c,m}$ for each row m . The weights of all columns and all rows are given by the *degree sequences*

$$\mathbf{d}_v = (d_{v,1}, \dots, d_{v,N}) \quad \text{and} \quad \mathbf{d}_c = (d_{c,1}, \dots, d_{c,M}). \quad (2.19)$$

The distribution of different column and row weights is described by a *degree distribution pair* in the following polynomial notation either in *edge perspective*

$$\lambda(x) = \sum_{i=2}^{d_v} \lambda_i x^{i-1} \quad \text{and} \quad \rho(x) = \sum_{j=2}^{d_c} \rho_j x^{j-1} \quad (2.20)$$

or in *node perspective*

$$\Lambda(x) = \sum_{i=2}^{d_v} \Lambda_i x^i \quad \text{and} \quad R(x) = \sum_{j=2}^{d_c} R_j x^j. \quad (2.21)$$

Here, d_v is the maximum column weight and d_c is the maximum row weight. The coefficients λ_i and ρ_j of the edge perspective describe the proportion of non-zero entries found in columns of weight i and in rows of weight j , respectively. The

coefficients Λ_i and R_j of the node perspective correspond to the proportion of columns of weight i and rows of weight j , respectively. Thus, $\lambda(x)$ and $\Lambda(x)$ are also called *column degree distribution* while $\rho(x)$ and $R(x)$ are called *row degree distribution*. A valid degree distribution has coefficients between zero and one and $\Lambda(1) = R(1) = \lambda(1) = \rho(1) = 1$ holds. Node and edge perspective are related by

$$\lambda(x) = \frac{\frac{d}{dx}\Lambda(x)}{\frac{d}{dx}\Lambda(x)|_{x=1}} = \frac{\Lambda'(x)}{\Lambda'(1)} \quad \text{and} \quad \rho(x) = \frac{\frac{d}{dx}R(x)}{\frac{d}{dx}R(x)|_{x=1}} = \frac{R'(x)}{R'(1)}. \quad (2.22)$$

The connection between the code rate and the degree distribution pair is given by

$$r = 1 - \frac{\sum_{i=2}^{d_v} i\Lambda_i}{\sum_{j=2}^{d_c} jR_j} = 1 - \frac{\sum_{j=2}^{d_c} \rho_j/j}{\sum_{i=2}^{d_v} \lambda_i/i} = 1 - \frac{\bar{d}_v}{\bar{d}_c}, \quad (2.23)$$

with $\bar{d}_v$ and $\bar{d}_c$ denoting the mean column and row weights. A regular LDPC code is a special case of irregular LDPC codes with $\lambda(x) = x^{d_v-1}$ and $\rho(x) = x^{d_c-1}$.

The so-called (λ, ρ) *ensemble* contains all LDPC codes of arbitrary length and with any (randomly constructed) check matrix that has column degree distribution $\lambda(x)$ and row degree distribution $\rho(x)$. The more restricted (N, λ, ρ) ensemble contains only those LDPC codes from the (λ, ρ) ensemble that have length N and thus check matrices of dimension $N(1-r) \times N$ with r as given by (2.23). The notion of an LDPC code ensemble is of particular interest, since the average error correction performance of an ensemble can be determined (semi-) analytically (for $N \rightarrow \infty$) [RU01b, RSU01] (cf. Section 2.7). The error correction capabilities of a concrete code realization of finite length, on the other hand, are much harder to predict, and except for special cases they can only be determined by time-consuming Monte Carlo methods.

2.3 Tanner Graph Description

A representation of LDPC codes by *bipartite* graphs was introduced in [Tan81]. Today, this representation is often called *Tanner graph* and is widely used especially to describe and analyze most LDPC decoding algorithms. Being bipartite, a Tanner graph $\mathcal{G} = (\mathcal{V} \cup \mathcal{C}, \mathcal{E})$ consists of the two disjunct sets $\mathcal{V}$ and $\mathcal{C}$ of N *variable nodes* and M *check nodes*, respectively, as well as the set $\mathcal{E}$ of edges with each edge connecting one variable node to one check node. The n -th variable node corresponds to the n -th check matrix column and represents one code word symbol y_n . The m -th check node corresponds to the m -th check matrix row and represents the m -th check equation $\mathbf{y}\mathbf{H}_{m,*}^\top = 0$. Accordingly, an edge is placed between variable node n and⁵ check node m if and only if $H_{m,n} \neq 0$. For non-binary codes this edge is assigned a weight $H_{m,n} \in \mathbb{F}_q^+$, while for binary codes the edge weights are usually discarded since all weights are equal to one. For systematic codes with a code word structure

⁵To simplify notation, the same variable is used to denote the index of a node and the node itself. E.g., the n -th variable node is simply referred to as variable node n .

as in (2.8), the variable nodes $\mathcal{V} = \{\mathcal{I} \cup \mathcal{P}\}$ can be further partitioned into K *information nodes* $\mathcal{I}$ and M *parity nodes* $\mathcal{P}$. In this form the first K columns of $\mathbf{H}$ and likewise the first K variable nodes correspond to the systematic information symbols, thus the name information nodes, while the remaining M columns and variable nodes correspond to the parity symbols, thus the name parity nodes. Below some properties of Tanner graphs are given (cf., e.g., [RL09]). An exemplary Tanner graph is depicted in Figure 2.2 on page 15 to illustrate these properties.

- **Node degree:** The *degree* of a node is defined as the number of incident edges. The degree $d_{v,n}$ of variable node n is equal to the weight of the n -th column of $\mathbf{H}$. The degree $d_{c,m}$ of check node m is equal to the weight of the m -th row.
- **Path:** A *path* of length l is defined by a sequence of $l + 1$ nodes, such that each pair of consecutive nodes is connected by an edge and no node appears more than once.
- **Cycle:** A *cycle* of length l is defined as a *closed path* of length l , starting and ending at the same node. In case of (bipartite) Tanner graphs, all cycles have even length due to the fact that the closed path defining a cycle is an alternating sequence of variable nodes and check nodes. Since Tanner graphs have no parallel edges (i.e., no two edges are incident to the same pair of nodes), the shortest possible cycle length is 4.
- **Girth:** The shortest cycle length in a graph is called its *girth* and is denoted g .
- **Distance:** The *distance* is defined as the length of the shortest path between two nodes. The distance from a node to itself is defined as zero.
- **Neighborhood of depth l :** For $l \in \mathbb{N}$, l odd, the neighborhood of *depth* l of variable node n is denoted $\mathcal{M}^l(n)$ and contains all check nodes m , such that the distance between variable node n and check node m is l . For l even, the neighborhood $\mathcal{N}^l(n)$ of *depth* l of variable node n contains all variable nodes n' , such that the distance between variable node n and variable node n' is l . These neighborhoods are computed recursively by:

$$\mathcal{N}^0(n) = \{n\}, \quad (2.24)$$

$$\mathcal{M}^1(n) = \mathcal{M}(n), \quad (2.25)$$

and for $l > 1$

$$\mathcal{N}^l(n) = \left\{ \bigcup_{m \in \mathcal{M}^{l-1}(n)} \mathcal{N}(m) \setminus \bigcup_{l' \in \{0,2,\dots,l-2\}} \mathcal{N}^{l'}(n) \right\}, \quad l \text{ even}, \quad (2.26)$$

$$\mathcal{M}^l(n) = \left\{ \bigcup_{n' \in \mathcal{N}^{l-1}(n)} \mathcal{M}(n') \setminus \bigcup_{l' \in \{1,3,\dots,l-2\}} \mathcal{M}^{l'}(n) \right\}, \quad l \text{ odd}. \quad (2.27)$$

The neighborhoods $\mathcal{M}^l(m)$ and $\mathcal{N}^l(m)$ of a check node m are defined in an analogous manner.

- **Tree up to depth l :** The *tree up to depth* l originating in variable node n

contains all variable and check nodes in the neighborhoods up to depth l :

$$\mathcal{T}_n^l = \left(\bigcup_{l' \in \{0,2,\dots,l\}} \mathcal{N}^{l'}(n) \right) \cup \left(\bigcup_{l' \in \{1,3,\dots,l\}} \mathcal{M}^{l'}(n) \right). \quad (2.28)$$

Again, an analogous definition holds for trees $\mathcal{T}_m^l$ originating in a check node m .

Cycles, Girth, and ACE of LDPC Codes

As will be illustrated in Section 2.5, the *cycle* structure of Tanner graphs has a substantial impact on the performance of *message-passing decoders*. In general, the shorter a cycle the more likely it is to deteriorate the decoding process as it inflicts statistical dependencies between exchanged messages. Some measures quantifying this impact are reviewed in the following.

- **Girth:** Undoubtedly, the most widely used measure – and at the same time the least informative one – is the girth g of a Tanner graph.⁶ Even though the minimum cycle length does not reveal any detailed cycle structure of a graph, in many cases it can still be assumed that most LDPC codes with higher girth will perform better than most codes with lower girth. Already in the original publication [Gal62], the role of the girth was realized and LDPC codes were proposed to be constructed with girth six or higher only.
- **Local girth:** Besides the global girth of a graph, also the *local girth* is of interest. The local girth g_n of each variable node n is defined as the length of the shortest cycle running through that node. The vector containing the local girths of all variable nodes is denoted $\mathbf{g} = (g_1, \dots, g_N)$, where $g = \min_{\forall n} g_n$ holds.
- **Cycle histogram:** A more detailed view than just the shortest cycle length is given by the *cycle histogram* up to length $l_{\max}$ denoted $\boldsymbol{\alpha} = (\alpha_4, \alpha_6, \dots, \alpha_{l_{\max}})$, where α_l corresponds to the total number of distinct cycles of length l that are found in the graph. The number of cycles in a Tanner graph quickly grows with the cycle length, and at the same time the influence of cycles on the decoding performance quickly diminishes with increasing length. Thus, the number of cycles is usually only of interest up to some maximum length. An algorithm to count the number of cycles of length g , $g+2$, and $g+4$ was proposed in [HC06]. Another algorithm with lower complexity that is capable of counting cycles of lengths from g up to $2g-2$ was presented in [KB10, KB13].
- **Local cycle histogram:** For each variable node n , the *local cycle histogram* $\boldsymbol{\alpha}_n = (\alpha_{n,4}, \alpha_{n,6}, \dots, \alpha_{n,l_{\max}})$ contains the number of cycles of length 4 to $l_{\max}$ running through that variable node.
- **ACE [TJVW04]:** The *approximate cycle extrinsic message degree* (ACE) of a cycle is $\sum_i (d_{v,i} - 2)$, where the sum is taken over all variable nodes participating

⁶Equivalently, an LDPC code is said to have a certain girth if its Tanner graph has that girth. The same also holds for other quantities that are technically graph – not code – properties.

in the cycle. The ACE provides an easily computable upper bound to the *extrinsic message degree* (EMD) of a cycle, which is defined as the number of check nodes that are singly connected to variable nodes of that cycle. Thus, the ACE provides a measure for the amount of extrinsic knowledge that is conveyed into a cycle by *message-passing decoding* (cf. Section 2.5), where large ACE values are advantageous as they represent high amounts of extrinsic information.

- **ACE spectrum** [VS09]: The *ACE spectrum* $\boldsymbol{\eta} = (\eta_4, \eta_6, \dots, \eta_{l_{\max}})$ is composed of the minimum ACE values η_l of all cycles of length l for cycle lengths from 4 to some maximum cycle length $l_{\max}$. If a graph does not have cycles of length l , the value η_l is set to ∞ .
- **Local ACE spectrum** [VS09]: For each variable node n , the *local ACE spectrum* $\boldsymbol{\eta}_n = (\eta_{n,4}, \dots, \eta_{n,l_{\max}})$ contains the minimum ACE values $\eta_{n,l}$ of all cycles of length l running through that variable node for cycle lengths from 4 to some maximum cycle length $l_{\max}$.
- **Expanded ACE spectrum** [VS09]: The *expanded ACE spectrum* Φ^{exp} is compiled by the numbers $\Phi_{\eta,l}^{\text{exp}}$ of variable nodes that have η as the minimum ACE value at cycle length l , i.e., for which $\eta_{n,l} = \eta$ holds:

$$\Phi^{\text{exp}} = \begin{pmatrix} \Phi_{0,4}^{\text{exp}} & \Phi_{0,6}^{\text{exp}} & \dots & \Phi_{0,l_{\max}}^{\text{exp}} \\ \Phi_{1,4}^{\text{exp}} & \Phi_{1,6}^{\text{exp}} & \dots & \Phi_{1,l_{\max}}^{\text{exp}} \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_{\eta_{\max},4}^{\text{exp}} & \Phi_{\eta_{\max},6}^{\text{exp}} & \dots & \Phi_{\eta_{\max},l_{\max}}^{\text{exp}} \end{pmatrix}, \text{ with } 0 \leq \Phi_{\eta,l}^{\text{exp}} \leq N. \quad (2.29)$$

The expanded ACE spectrum is only computed up to a maximum length $l_{\max}$ and up to a maximum ACE value $\eta_{\max}$.

- **ACE-cycle histogram**: The *ACE-cycle histogram* is compiled by the numbers $\Phi_{\eta,l}^{\text{cyc}}$ of cycles of length exactly l and with ACE exactly η and is represented by the matrix

$$\Phi^{\text{cyc}} = \begin{pmatrix} \Phi_{0,4}^{\text{cyc}} & \Phi_{0,6}^{\text{cyc}} & \dots & \Phi_{0,l_{\max}}^{\text{cyc}} \\ \Phi_{1,4}^{\text{cyc}} & \Phi_{1,6}^{\text{cyc}} & \dots & \Phi_{1,l_{\max}}^{\text{cyc}} \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_{\eta_{\max},4}^{\text{cyc}} & \Phi_{\eta_{\max},6}^{\text{cyc}} & \dots & \Phi_{\eta_{\max},l_{\max}}^{\text{cyc}} \end{pmatrix}. \quad (2.30)$$

As above, cycles are only counted up to a maximum length $l_{\max}$ and up to a maximum ACE value $\eta_{\max}$.

Knowing about the detrimental effect of cycles, the attempt of designing codes with cycle-free graphs seems reasonable. However, it was shown in [ETV99] that cycle-free graphs cannot support good codes, meaning that they have very low minimum Hamming distance and also are very likely to perform poorly in the waterfall region. Moreover, it was shown that to support good codes, the number of cycles in a graph even needs to grow exponentially with the code length.

Two construction methods for check matrices that are based on the cycle properties of LDPC codes are given in Section 2.8 on pages 38 ff.

Exemplary LDPC Code

To illustrate some of the properties introduced above, the following exemplary check matrix of dimension 9×12 is considered:

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}. \quad (2.31)$$

The corresponding Tanner graph in Figure 2.2 has $M = 9$ check nodes and $N = 12$ variable nodes, where the latter ones are composed of $K = 3$ information and $M = 9$ parity nodes. Since the check matrix has full rank, the code rate equals $r = \frac{12-9}{12} = \frac{1}{4}$. The neighborhoods $\mathcal{M}(1)$ and $\mathcal{N}(1)$ of depth 1 are indicated by the dotted lines for the first variable and check node. The degree distribution pair is given by

$$\begin{aligned} \lambda(x) &= \frac{14}{29}x + \frac{15}{29}x^2, & \rho(x) &= \frac{21}{29}x^2 + \frac{8}{29}x^3 \quad (\text{edge perspective}) \\ \text{or } \Lambda(x) &= \frac{7}{12}x^2 + \frac{5}{12}x^3, & R(x) &= \frac{7}{9}x^3 + \frac{2}{9}x^4 \quad (\text{node perspective}). \end{aligned}$$

The bold line shows a cycle of length 6 with ACE $(d_{v,3}-2) + (d_{v,6}-2) + (d_{v,7}-2) = 2$. The local girth of all variable nodes, the cycle histogram up to length $l_{\max} = 10$, the

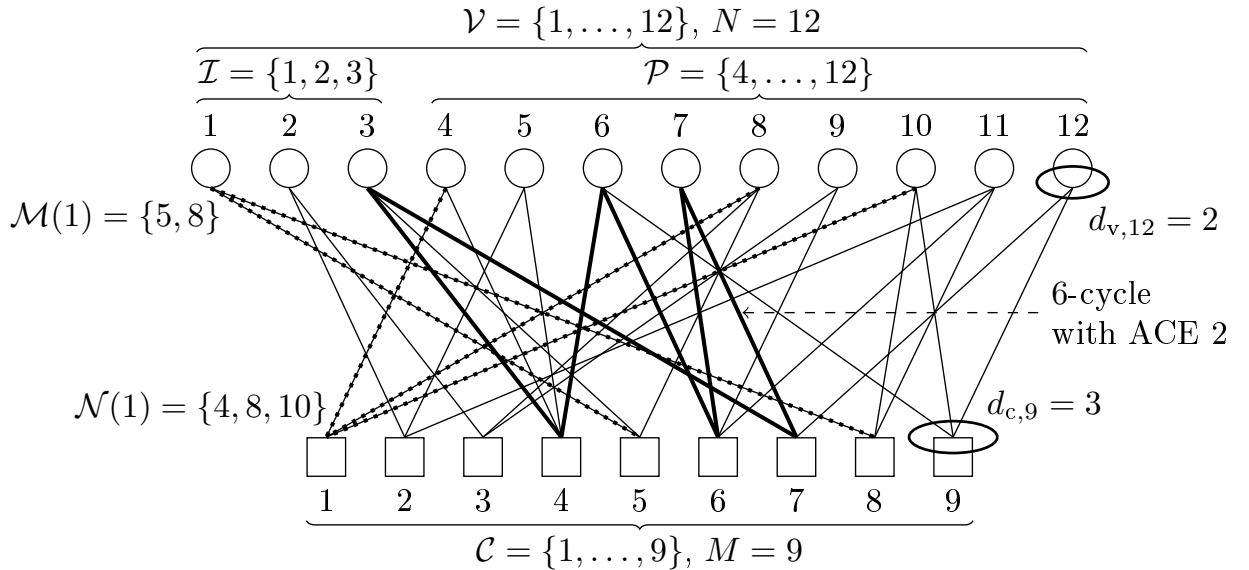


Figure 2.2: Tanner graph of LDPC code example.

ACE spectrum up to length $l_{\max} = 10$, the expanded ACE spectrum up to length $l_{\max} = 10$ and up to ACE $\eta_{\max} = 2$, as well as the ACE-cycle histogram up to length $l_{\max} = 10$ and up to ACE $\eta_{\max} = 5$ are given below:

$$\begin{aligned}
\mathbf{g} &= (g_1, \dots, g_{12}) = (6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6) && \text{(local girths),} \\
\boldsymbol{\alpha} &= (\alpha_4, \alpha_6, \alpha_8, \alpha_{10}) = (0, 9, 25, 34) && \text{(cycle histogram),} \\
\boldsymbol{\eta} &= (\eta_4, \eta_6, \eta_8, \eta_{10}) = (\infty, 1, 1, 1) && \text{(ACE spectrum),} \\
\boldsymbol{\Phi}^{\text{exp}} &= \begin{pmatrix} \Phi_{0,4}^{\text{exp}} & \cdots & \Phi_{0,10}^{\text{exp}} \\ \vdots & \ddots & \vdots \\ \Phi_{2,4}^{\text{exp}} & \cdots & \Phi_{2,10}^{\text{exp}} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 6 & 9 & 8 \\ 0 & 6 & 3 & 4 \end{pmatrix} && \text{(exp. ACE spectrum),} \\
\boldsymbol{\Phi}^{\text{cyc}} &= \begin{pmatrix} \Phi_{0,4}^{\text{cyc}} & \cdots & \Phi_{0,10}^{\text{cyc}} \\ \vdots & \ddots & \vdots \\ \Phi_{5,4}^{\text{cyc}} & \cdots & \Phi_{5,10}^{\text{cyc}} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 1 \\ 0 & 6 & 13 & 6 \\ 0 & 1 & 9 & 18 \\ 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 1 \end{pmatrix} && \text{(ACE-cycle histogram).}
\end{aligned}$$

2.4 Encoding

The encoding complexity is a major concern when implementing LDPC codes, as in the most general case the execution time scales quadratically with the code length, i.e., it lies in $\mathcal{O}(N^2)$. For increasing code lengths this quickly becomes computationally infeasible and alternative approaches that require special code structures have to be employed. The encoding methods described in this section are applicable to both binary and non-binary LDPC codes.

Multiplication with the Generator Matrix

In general, for any LDPC code defined by its (full rank) check matrix $\mathbf{H}$, the systematic generator matrix $\mathbf{G}$ can be uniquely determined. With the systematic representation (2.7), the expression $\mathbf{G} \cdot \mathbf{H}^T \stackrel{(2.10)}{=} \mathbf{0}$ can be rephrased to

$$\begin{aligned}
&(\mathbf{I}_K, \mathbf{G}_{[K+1 \rightarrow N]}) \cdot (\mathbf{H}_{[1 \rightarrow K]}, \mathbf{H}_{[K+1 \rightarrow N]})^T = \mathbf{0} \\
\Leftrightarrow &\mathbf{G}_{[K+1 \rightarrow N]} \cdot \mathbf{H}_{[K+1 \rightarrow N]}^T = \mathbf{H}_{[1 \rightarrow K]}^T.
\end{aligned} \tag{2.32}$$

For this system of linear equations with $K \cdot M$ unknown entries in $\mathbf{G}$ and $K \cdot M$ equations to be uniquely solvable, $\mathbf{H}_{[K+1 \rightarrow N]}$ is required to have full rank. Consequently, for systematic encoding the condition of $\mathbf{H}$ having full rank is w.l.o.g. further confined to the rightmost M columns of $\mathbf{H}$ being linearly independent. Having computed the parity part $\mathbf{G}_{[K+1 \rightarrow N]}$ of the generator matrix, encoding is achieved by copying the information word $\mathbf{x}$ to the first K positions of the code word $\mathbf{y}$ and filling the remaining M positions with the parity word $\mathbf{p}$ which is computed by

$\mathbf{p} = \mathbf{x}\mathbf{G}_{[K+1 \rightarrow N]}$. While being generally applicable, this method has the major drawback of quadratic memory and time complexity: storing the generator matrix requires $\mathcal{O}(KM) = \mathcal{O}(r(1-r)N^2) = \mathcal{O}(N^2)$ memory elements and computing the parity symbols requires $\mathcal{O}(KM) = \mathcal{O}(r(1-r)N^2) = \mathcal{O}(N^2)$ operations.

Approximate Triangulation of the Check Matrix [RU01a]

An efficient encoding algorithm that is generally applicable to LDPC codes was introduced in [RU01a]. Merely by reordering columns and rows – and thus not changing the density – of the check matrix $\mathbf{H}$, the $M \times M$ parity part $\mathbf{H}_{[K+1 \rightarrow N]}$ is transformed into *approximate upper triangular* form $\mathbf{H}_{[K+1 \rightarrow N]} = \begin{pmatrix} \mathbf{T} & \mathbf{A} \\ \mathbf{E} & \mathbf{C} \end{pmatrix}$, with $\mathbf{T}$ being an upper triangular matrix of dimension $(M - g) \times (M - g)$ with non-zero entries on the main diagonal, and arbitrary structure of $\mathbf{A}$, $\mathbf{E}$, and $\mathbf{C}$. With this modified representation, encoding can be accomplished with time complexity $\mathcal{O}(N + g^2)$. By using a greedy algorithm for the approximate triangulation of the check matrix, a small size g of the so-called *gap* can be achieved, resulting in quasi linear encoding complexity. Details on the greedy approximate triangulation and the subsequent encoding process can be found in [RU01a, RU08].

Novel Approximate Triangulation for Encoding without Column Swaps

A drawback of encoding by approximate triangulation as proposed in [RU01a] is that the greedy algorithm swaps many columns and rows in the check matrix to achieve a small gap g . The swapping of columns coincides with a corresponding reordering of the code symbol positions. For codes that have been specifically optimized to have a certain order of columns, this reordering is undesirable. One reason for the importance of the column order can be specific roles that are assigned to specific code symbol positions, like certain symbols being candidates for adapting the code rate by *shortening* or *puncturing* (see Chapter 3) or different symbols having *unequal error protection* (UEP) levels. Another reason can be decoder architectures using sophisticated *message-passing schedules* (see Chapter 5) which perform message updates in specifically optimized orders, and thus being susceptible to changes in the order of check matrix columns as well as rows. To account for these effects, an alternative approximate triangulation and encoding method is proposed in the following.

The central requirement of this method is that the original check matrix $\mathbf{H}$ can be used for decoding without modifications. This is achieved by finding a reordered matrix $\tilde{\mathbf{H}}$ which enables efficient encoding and subsequently applying a permutation to the resulting code word to ensure compatibility with the original matrix $\mathbf{H}$. For systematic encoding, the parity check equations (2.11) can be rephrased as follows:

$$\begin{aligned} & (\mathbf{x}, \mathbf{p}) \cdot (\mathbf{H}_{[1 \rightarrow K]}, \mathbf{H}_{[K+1 \rightarrow N]})^\top = \mathbf{0} \\ \Leftrightarrow & \mathbf{p} \cdot \mathbf{H}_{[K+1 \rightarrow N]}^\top = \mathbf{x} \cdot \mathbf{H}_{[1 \rightarrow K]}^\top \end{aligned} \quad (2.33)$$

$$\Leftrightarrow \mathbf{p} = \mathbf{x} \cdot \mathbf{H}_{[1 \rightarrow K]}^\top \cdot \left(\mathbf{H}_{[K+1 \rightarrow N]}^\top \right)^{-1}. \quad (2.34)$$

Using (2.33), encoding – which means finding $\mathbf{p}$ for a given $\mathbf{x}$ – could be done in linear time by back substitution if the square full rank matrix $\mathbf{H}_{[K+1 \rightarrow N]}$ had upper triangular form, which is not the case in general. To approximate this triangular form, the modified check matrix

$$\tilde{\mathbf{H}} = \mathbf{P}_r \cdot (\mathbf{H}_{[1 \rightarrow K]}, \mathbf{H}_{[K+1 \rightarrow N]} \cdot \mathbf{P}_c) = (\mathbf{P}_r \cdot \mathbf{H}_{[1 \rightarrow K]}, \mathbf{P}_r \cdot \mathbf{H}_{[K+1 \rightarrow N]} \cdot \mathbf{P}_c) \quad (2.35)$$

is constructed by arbitrarily exchanging rows and exchanging columns only within the parity part of $\mathbf{H}$ such that $\mathbf{P}_r \cdot \mathbf{H}_{[K+1 \rightarrow N]} \cdot \mathbf{P}_c$ is in approximate upper triangular form, where the gap size g is as small as possible. The $M \times M$ permutation matrix $\mathbf{P}_r$ describes the row exchanges while the $M \times M$ permutation matrix $\mathbf{P}_c$ describes the column exchanges within the parity part.⁷ In contrast to the approach of [RU01a], here no exchanges of columns between the information part and parity part are allowed to ensure that the information symbol positions are not changed. This is achieved by using the greedy algorithm in [RU08, Appendix A.2] in a modified version that only chooses columns in the range $K + 1$ to N . Although, the resulting gap g is generally larger than for the original greedy triangulation, empirical observations show that in many cases $g \ll M$ still holds and near linear time encoding can thus be achieved. Using the modified check matrix $\tilde{\mathbf{H}}$ and (2.33), encoding results in the code word $\tilde{\mathbf{y}} = (\mathbf{x}, \tilde{\mathbf{p}})$ with

$$\begin{aligned} \tilde{\mathbf{p}} \cdot \tilde{\mathbf{H}}_{[K+1 \rightarrow N]}^\top &= \mathbf{x} \cdot \tilde{\mathbf{H}}_{[1 \rightarrow K]}^\top \\ \Leftrightarrow \tilde{\mathbf{p}} \cdot (\mathbf{P}_r \cdot \mathbf{H}_{[K+1 \rightarrow N]} \cdot \mathbf{P}_c)^\top &= \mathbf{x} \cdot (\mathbf{P}_r \cdot \mathbf{H}_{[1 \rightarrow K]})^\top \\ \Leftrightarrow \tilde{\mathbf{p}} \cdot \mathbf{P}_c^\top \cdot \mathbf{H}_{[K+1 \rightarrow N]}^\top \cdot \mathbf{P}_r^\top &= \mathbf{x} \cdot \mathbf{H}_{[1 \rightarrow K]}^\top \cdot \mathbf{P}_r^\top \\ \Leftrightarrow \tilde{\mathbf{p}} \cdot \mathbf{P}_c^\top &= \mathbf{x} \cdot \mathbf{H}_{[1 \rightarrow K]}^\top \cdot (\mathbf{H}_{[K+1 \rightarrow N]}^\top)^{-1} \\ \Rightarrow \mathbf{p} = \tilde{\mathbf{p}} \cdot \mathbf{P}_c^\top &= \tilde{\mathbf{p}} \cdot \mathbf{P}_c^{-1}. \end{aligned} \quad (2.36)$$

It follows, that the desired code word $\mathbf{y} = (\mathbf{x}, \mathbf{p})$ can be easily computed from $\tilde{\mathbf{y}}$ by applying the inverse permutation $\mathbf{P}_c^{-1}$ to the parity word $\tilde{\mathbf{p}}$. Using this method, efficient encoding can be realized by using the modified matrix $\tilde{\mathbf{H}}$ and the permutation $\mathbf{P}_c^{-1}$, while the decoder can use the matrix $\mathbf{H}$ without any modifications.

Structured LDPC Codes

Although the methods described above are generally applicable to LDPC codes, in practical systems even more efficient encoders are desirable that do not have to store a specific encoding matrix (i.e., neither the generator matrix nor a triangulated version

⁷A permutation matrix is defined as a binary square matrix that has exactly one 1-entry in each row and column and zeros elsewhere. When a matrix $\mathbf{M}$ is multiplied by a permutation matrix from the left, a 1 in the i -th row and j -th column of the permutation matrix moves the j -th row of $\mathbf{M}$ to the i -th position. On the other hand, when a matrix $\mathbf{M}$ is multiplied by a permutation matrix from the right, a 1 in the i -th row and j -th column of the permutation matrix moves the i -th column of $\mathbf{M}$ to the j -th position. Furthermore, the inverse of a permutation matrix equals its transpose, i.e., $\mathbf{P}^{-1} = \mathbf{P}^\top$.

of the check matrix). This is commonly achieved by using LDPC codes with specially structured check matrices for which specific linear time encoders are available. The most prominent examples include:

- *irregular repeat-accumulate* (IRA) codes [JKM00] as used, e.g., in the *digital video broadcast* standards DVB-S2 [Eur09a] for satellite and DVB-T2 [Eur09b] for terrestrial transmission,
- *quasi-cyclic* (QC) LDPC codes (see Section 2.9) as used, e.g., in the wireless standards IEEE 802.11n (WLAN) [IEE09b] and IEEE 802.16e (WiMAX) [IEE09a],
- and *LDPC convolutional* codes [JFZ99] as used, e.g., in IEEE 1901-2010 (broadband over power lines) [IEE10].

2.5 Message-Passing Decoding

It was shown in [Gal63] that the minimum distance of randomly constructed LDPC codes grows linearly with increasing code length if the check matrix is sufficiently sparse. Furthermore, it was proved in [Mac99] that the Shannon capacity [Sha48] can be approached arbitrarily closely with LDPC codes if the code length goes to infinity and optimal *maximum-likelihood* (ML) decoding is applied. While this property has also been shown for other codes before, the reason for the success of LDPC codes is the availability of decoding algorithms with complexity growing only linear with the code length. The most widely used LDPC decoding algorithms belong to the class of *message-passing* algorithms which iteratively compute and pass messages between nodes in the graph representation of a code. Unfortunately, these decoding algorithms are suboptimal in the sense that they not always yield the ML solution. This suboptimality is inflicted by cycles within the graph, which cause messages to transport statistically dependent information after some iterations. Nevertheless, empirical results indicate that even under suboptimal message-passing decoding, error correction performance very close to the theoretical Shannon capacity can be achieved [CFRU01] with linear complexity, whereas generally ML decoding has exponential complexity [Var97]. A brief description of message-passing decoding in general and the prevalent instance of the *belief propagation* (BP) (or *sum-product*) decoder are presented in the following.

Given the bipartite Tanner graph description of an LDPC code, a message-passing decoder is determined by four main components.

1. The *message generating function* $f_{v \rightarrow c}(\cdot)$ that computes messages $v_{n,m}$ sent from variable node n to check node m :

$$v_{n,m} = f_{v \rightarrow c} \left(c_{m_1,n}, \dots, c_{m_{d_v,n},n} \right) \quad \forall n \in \{1, \dots, N\}, \quad m_i \in \mathcal{M}(n). \quad (2.37)$$

2. The message generating function $f_{c \rightarrow v}(\cdot)$ that computes messages $c_{m,n}$ sent from check node m to variable node n :

$$c_{m,n} = f_{c \rightarrow v} \left(v_{n_1,m}, \dots, v_{n_{d_c,m},m} \right) \quad \forall m \in \{1, \dots, M\}, n_i \in \mathcal{N}(m). \quad (2.38)$$

3. A *message-passing schedule* that determines in which order messages are computed and passed to neighboring nodes.
4. A *stopping rule* that is used to decide when to finalize the message-passing process.

A tremendous variety of message-passing algorithms differing in all four components has been proposed for the decoding of LDPC codes, e.g., [FMI99], [MB01a], [ZF02], [Hoc04], [CGW07a], [YEK⁺07]. Nevertheless, only the instance of belief propagation decoding in the LLR domain for binary LDPC codes is introduced here. For the case of non-binary codes over $\mathbb{F}_q$, a generalized, though more complex, decoder is given in Section 2.10. Furthermore, a more extensive analysis of decoders with different message-passing schedules is conducted in Chapter 5.

Belief Propagation Decoding for Binary LDPC Codes

As only transmission over the AWGN channel is considered, BP decoding always starts by initializing all variable-to-check messages $v_{n,m}$ with the channel LLRs L_n :

$$v_{n,m} = L_n = 4 \frac{E_s}{N_0} z_n, \quad \forall n \in \{1, \dots, N\}, m \in \mathcal{M}(n). \quad (2.39)$$

Then, in the so-called *check node update* (or *horizontal step*) the outgoing messages of all check nodes are computed using the message generating function $f_{c \rightarrow v}$:

$$\begin{aligned} c_{m,n} &= f_{c \rightarrow v} \left(v_{n_1,m}, \dots, v_{n_{d_c,m},m} \right) \\ &= 2 \operatorname{atanh} \left(\prod_{n' \in \mathcal{N}(m) \setminus n} \tanh \left(\frac{v_{n',m}}{2} \right) \right) \end{aligned} \quad (2.40)$$

$$= \boxplus_{n' \in \mathcal{N}(m) \setminus n} v_{n',m} \quad \forall m \in \{1, \dots, M\}, n \in \mathcal{N}(m), \quad (2.41)$$

where the operator $\setminus$ denotes exclusion from a set and $\boxplus$ represents a summation using the *boxplus* operator $\boxplus$ [HOP96]. Thus, an outgoing message of a check node represents the extrinsic (boxplus) summation of all incoming messages of that check node, except for the one that came from the same edge the message is sent to. In the subsequent *variable node update* (also *vertical step*) the outgoing messages of all variable nodes are computed using the message generating function $f_{v \rightarrow c}$:

$$\begin{aligned} v_{n,m} &= f_{v \rightarrow c} \left(c_{m_1,n}, \dots, c_{m_{d_v,n},n} \right) \\ &= L_n + \sum_{m' \in \mathcal{M}(n) \setminus m} c_{m',n} \quad \forall n \in \{1, \dots, N\}, m \in \mathcal{M}(n). \end{aligned} \quad (2.42)$$

An outgoing message of a variable node is computed by adding the channel LLR L_n and the extrinsic knowledge of all incoming messages, except for the one that came from the same edge the message is sent to. This summation is motivated by the assumption that all incoming messages are statistically independent. In general, this assumption only holds for cycle free graphs. As mentioned above, LDPC codes do not have cycle free graphs, but rather exhibit a large number of cycle. This causes inaccuracies in the message updates since after only a few iterations statistically dependent knowledge is summed up. The exact tracking of all statistical dependencies of messages is prohibitively complex such that the resulting inaccuracies have to be tolerated.

The default message-passing schedule used in BP decoding is the so-called *flooding* schedule. In the check node update all check-to-variable messages are computed in parallel and propagated (*flooded*) to the connected variable nodes. Likewise, in the subsequent variable node update all variable-to-check messages are computed in parallel and propagated to the connected check nodes. One iteration of the flooding schedule, thus, consists of the computation and propagation of $|\mathbf{H}|$ check-to-variable messages followed by the computation and propagation of $|\mathbf{H}|$ variable-to-check messages.

The default stopping rule consists of two criteria. Firstly, after each iteration the parity check equations are evaluated using a hard decision

$$\hat{y}_n = \begin{cases} 0 & \text{if } v_n \geq 0 \\ 1 & \text{if } v_n < 0 \end{cases}, \quad \forall n \in \{1, \dots, N\} \quad (2.43)$$

computed from the current *a posteriori* LLRs

$$v_n = L_n + \sum_{m \in \mathcal{M}(n)} c_{m,n} \quad \forall n \in \{1, \dots, N\}, \quad (2.44)$$

which contain the channel knowledge L_n and all extrinsic information gathered from neighboring check nodes. If all check equations are fulfilled, decoding is stopped by declaring a decoding success, and the estimates $\hat{y}_1, \dots, \hat{y}_K$ of the transmitted information symbols are forwarded. If, on the other hand, no valid code word is found after a predefined maximum number $I_{\max}$ of flooding iterations, decoding is aborted by declaring a decoding failure and, depending on the application, the (erroneous) estimates $\hat{y}_1, \dots, \hat{y}_K$ can be forwarded to the next block in the receiver chain. In summary, there are three possible outcomes of the decoding process:

1. No code word is found:
Errors are still present in the decoder output and the decoder declares a failure.
2. The correct code word is found:
The decoder declares a success and there are no errors in the output.
3. A wrong code word is found:
The decoder declares a success but there are still errors in the output. These errors are called *undetected errors*.

2.6 Performance Measures

Throughout this thesis, several measures are used to evaluate the performance of LDPC code ensembles and of specific code instances regarding error correction capabilities, convergence speed of iterative decoding, and computational complexity.

2.6.1 Ensemble Performance

It is well-known that the degree distributions $\lambda(x)$ and $\rho(x)$ of an LDPC code strongly influence its error correcting capabilities [LMSS01, RSU01]. To evaluate a degree distribution pair, the asymptotic behavior of message-passing decoding of the (N, λ, ρ) ensemble containing all LDPC codes of length N with degree distributions $\lambda(x)$ and $\rho(x)$ is examined.

Ensemble Threshold

Under the assumption of infinite code length $N \rightarrow \infty$ and a cycle-free Tanner graph, a so-called *ensemble threshold* $\tau(\lambda, \rho) = (E_b/N_0)_{\text{thresh}}$ exists that represents the lowest channel quality $(E_b/N_0)_{\text{thresh}}$ at which error-free decoding is theoretically possible with an LDPC code from the (λ, ρ) ensemble [RU01b]. Different methods to compute this ensemble threshold are discussed in Section 2.7. In this sense, a code with a certain degree distribution pair (λ, ρ) can on average be expected to have better error correction properties than a code with a different degree distribution pair $(\tilde{\lambda}, \tilde{\rho})$ if the corresponding ensemble threshold $\tau(\lambda, \rho)$ is lower than $\tau(\tilde{\lambda}, \tilde{\rho})$.

Gap of Ensemble Threshold to Capacity

For the BIAWGN channel, a minimum channel quality $\mathbf{q}(r) = (E_b/N_0)_{r, \min}$ for which error free decoding is theoretically possible with a channel code of rate r can be computed from the Shannon capacity [Sha48] by restricting the channel input to binary values (see, e.g., [Bre14, Appendix A.3]). For an easier comparison of the performance of codes of different code rates, the gap of the ensemble threshold $\tau(\lambda, \rho)$ to the respective capacity limit $\mathbf{q}(r)$ is employed as performance indicator and denoted

$$\gamma(\lambda, \rho)|_{\text{dB}} = \tau(\lambda, \rho)|_{\text{dB}} - \mathbf{q}(r)|_{\text{dB}}, \quad (2.45)$$

where the notation $x|_{\text{dB}} = 10 \log_{10}(x)$ is used to convert a value to the dB scale. The resulting linear expression is thus given by

$$\gamma(\lambda, \rho) = \frac{\tau(\lambda, \rho)}{\mathbf{q}(r)}. \quad (2.46)$$

Consequently, a gap of the ensemble threshold to capacity of, e.g., 3 dB means that a code from the considered ensemble needs at least two times more energy for error-free transmission than a perfectly capacity-achieving code. It should be noted that ensemble thresholds as well as gaps to capacity will be mostly given on the dB scale if presented as numerical results.

Stability Condition

Given a degree distribution pair (λ, ρ) , the so-called *stability condition* [RSU01] provides a lower bound on the ensemble threshold, that is much easier to compute than the exact threshold and thus is commonly used as additional constraint during the optimization of degree distributions. For the BIAWGN channel the stability condition is given by [RSU01]

$$\lambda'(0)\rho'(1) < e^{\frac{1}{2\sigma_n^2}}, \quad (2.47)$$

where $\lambda'(x) = \frac{d}{dx}\lambda(x)$ and $\rho'(x) = \frac{d}{dx}\rho(x)$ denote the derivatives with respect to x and σ_n^2 denotes the AWGN channel noise variance which for code rate r is given by

$$\sigma_n^2 = \frac{N_0}{2rE_b} = \frac{1}{2rE_b/N_0} \stackrel{\uparrow}{=} \frac{1}{2r\tau(\lambda, \rho)} \quad \text{at threshold} \quad (2.48)$$

By inserting the resulting noise variance (2.48) at the ensemble threshold, the stability condition (2.47) can be rewritten as

$$\begin{aligned} \lambda'(0)\rho'(1) &< e^{r\tau(\lambda, \rho)} \\ \Leftrightarrow \quad \tau(\lambda, \rho) &> \frac{\ln(\lambda'(0)\rho'(1))}{r}, \end{aligned} \quad (2.49)$$

yielding a lower bound on the ensemble threshold. Another formulation of the stability condition is often used to upper bound the fraction of degree 2 variable nodes during the optimization of degree distributions:

$$\begin{aligned} \lambda'(0)\rho'(1) &= \lambda_2 \sum_{j=2}^{d_c} \rho_j(j-1) < e^{r\tau(\lambda, \rho)} \\ \Leftrightarrow \quad \lambda_2 &< \frac{e^{r\tau(\lambda, \rho)}}{\sum_{j=2}^{d_c} \rho_j(j-1)}. \end{aligned} \quad (2.50)$$

2.6.2 Code Performance

Once a specific code has been chosen from an ensemble, i.e., a check matrix $\mathbf{H}$ has been constructed, its performance is evaluated by Monte Carlo simulations.

Bit Error Rate (BER)

The *bit error rate* (BER) is measured by transmitting and decoding a sufficient number of random information words and observing the fraction of erroneous bits at the decoder output. The measured BER is, thus, an estimate of the *bit error probability* of the transmission system including forward error correction. In case of non-binary codes, the decoded symbols are expanded to their binary representation before the BER is measured.

Frame Error Rate (FER)

The *frame error rate* (FER) or *word error rate* is measured by transmitting and decoding a sufficient number of random information words and observing the fraction of decoded words that contain at least one error at the decoder output. Again, the true *frame error probability* of the transmission system including forward error correction is estimated by the FER.

Undetected BER (uBER) and undetected FER (uFER)

Whenever the decoder converges to a wrong code word, the resulting errors are counted as *undetected bit errors* and *undetected frame errors*, respectively. The relations

$$\text{uBER} \leq \text{BER} \quad \text{and} \quad (2.51)$$

$$\text{uFER} \leq \text{FER} \quad (2.52)$$

hold, since all undetected errors are also included in the overall error rates.

BER Threshold

In some cases, rather than the shape of the BER versus channel quality curve, it is only of interest at which channel quality a certain target BER is achieved by a code. This channel quality is referred to as *BER threshold* and the corresponding E_b/N_0 value is denoted $\tau^{[\text{BER}]}$. Due to a finite code length, the BER threshold is lower bounded by the respective ensemble threshold:

$$\tau^{[\text{BER}]} > \tau(\lambda, \rho). \quad (2.53)$$

Gap of BER Threshold to Capacity

Equivalently to the ensemble threshold, the gap of the BER threshold $\tau^{[\text{BER}]}$ to the respective capacity limit $\mathbf{q}(r)$ is employed for an easier comparison of codes with different code rates and denoted

$$\gamma^{[\text{BER}]} \Big|_{\text{dB}} = \tau^{[\text{BER}]} \Big|_{\text{dB}} - \mathbf{q}(r) \Big|_{\text{dB}} \quad \Leftrightarrow \quad \gamma^{[\text{BER}]} = \frac{\tau^{[\text{BER}]}}{\mathbf{q}(r)}. \quad (2.54)$$

Again, the gap of the BER threshold is lower bounded by the gap of the ensemble threshold:

$$\gamma^{[\text{BER}]} > \gamma(\lambda, \rho). \quad (2.55)$$

Average Number of Iterations / Message Updates

While message-passing decoders are usually constrained by the *maximum number of decoding iterations* $I_{\max}$, the average computational complexity is governed by the *average number of decoding iterations* I_{avg} that are executed at different channel qualities. For a fair comparison of cases in which the complexity per iteration is not constant, the average number U_{avg} of messages that are updated in total is employed instead.

Convergence Speed

Especially for low-complexity applications, the BER performance after only a few iterations is of interest. To evaluate the convergence speed of decoding for different codes, the BER or FER is measured at the end of each iteration. This way, the behavior of the error rate at different stages of the decoding process can easily be compared for different codes and/or different decoders.

2.7 Ensemble Optimization

The optimization of LDPC codes is commonly divided into two parts: finding a degree distribution pair (λ, ρ) with a sufficiently good ensemble threshold $\tau(\lambda, \rho)$, followed by finding a code instance of desired length and with sufficiently good error correction performance from this ensemble (i.e., constructing a check matrix $\mathbf{H}$ of corresponding size). The first part is referred to as *ensemble optimization* and covered for the case of binary LDPC codes in this section, while the second part is referred to as *code construction* which is covered for the case of binary LDPC codes in Section 2.8.

Before the ensemble optimization process is explained in Section 2.7.2, different methods of computing the ensemble threshold $\tau(\lambda, \rho)$ are presented in Section 2.7.1, since they are needed to evaluate the objective function of the optimization.

2.7.1 Density Evolution for Binary LDPC Codes

In [RU01b] *density evolution* (DE) was introduced as a technique for computing the ensemble thresholds of LDPC code ensembles. It is based on the idea of asymptotically tracking the probability density functions (pdfs) of the messages that are computed by a message-passing decoder to evaluate if it potentially converges to zero error probability. This process is briefly reviewed for the BIAWGN channel and BP decoding of binary LDPC codes in the following and is depicted in Figure 2.3.

Due to the linearity of LDPC codes, w.l.o.g. only the transmission of the all-zero code word is considered. With $E_s = 1$, the channel output LLRs can then be described by a Gaussian random variable with mean $\mu_L = 4\frac{E_s}{N_0}$, variance $\sigma_L^2 = 2\mu_L = 8\frac{E_s}{N_0}$, and pdf [tB01]

$$\begin{aligned} p_{\text{AWGN,LLR}}(x) &= \frac{1}{\sqrt{2\pi\sigma_L^2}} \exp\left(-\frac{(x - \mu_L)^2}{2\sigma_L^2}\right) \\ &= \frac{1}{4\sqrt{\pi\frac{E_s}{N_0}}} \exp\left(-\frac{\left(x - 4\frac{E_s}{N_0}\right)^2}{16\frac{E_s}{N_0}}\right). \end{aligned} \quad (2.56)$$

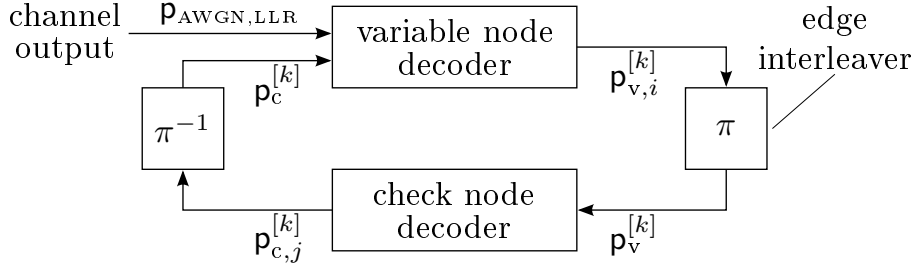


Figure 2.3: Block diagram of message-passing decoding with corresponding pdfs.

In the initialization step (2.39) of the BP decoder, all variable-to-check messages are initialized with the channel output LLRs. Thus, the initial pdf of these messages is

$$\mathbf{p}_v^{[0]} = \mathbf{p}_{\text{AWGN,LLR}}. \quad (2.57)$$

To simplify the notation, pdfs are written without their argument, e.g., $\mathbf{p}_v^{[0]} = \mathbf{p}_v^{[0]}(x)$.

Deriving the pdf $\mathbf{p}_{c,j}^{[k]}$ of an outgoing message of a degree- j check node after the k -th iteration according to the BP check node update (2.41) is rather laborious and thus not explicated here. A detailed description is found in [RSU01] and can be stated as

$$\mathbf{p}_{c,j}^{[k]} = \Gamma^{-1} \left(\left(\Gamma \left(\mathbf{p}_v^{[k-1]} \right) \right)^{\otimes(j-1)} \right), \quad (2.58)$$

where the transformation Γ to a special logarithmic domain is chosen in such a way, that the boxplus operation of two transformed random variables results in the convolution of the transformed pdfs. The corresponding inverse transform is denoted Γ^{-1} . Finally, $(\cdot)^{\otimes(j-1)}$ denotes the $(j-1)$ -fold convolution resulting from the boxplus summation of $(j-1)$ incoming messages which is performed at a degree j check node. More precisely, $x^{\otimes 0} := \delta(x)$, $x^{\otimes 1} = x$, $x^{\otimes 2} = x \otimes x$, and so forth, where $\delta(x)$ denotes the *Dirac delta function*. The edge interleaving is assumed to be random, such that each edge is connected to a degree j check node with probability ρ_j . Thus, the pdf $\mathbf{p}_v^{[k]}$ at the input of the variable node decoder is given by

$$\mathbf{p}_c^{[k]} = \Gamma^{-1} \left(\sum_{j=2}^{d_c} \rho_j \cdot \left(\Gamma \left(\mathbf{p}_v^{[k-1]} \right) \right)^{\otimes(j-1)} \right) =: \Gamma^{-1} \left(\rho \left(\Gamma \left(\mathbf{p}_v^{[k-1]} \right) \right) \right). \quad (2.59)$$

The variable node decoder assumes all incoming messages to be statistically independent and computes the outgoing messages of a degree i variable node as the summation of its channel LLR and $(i-1)$ incoming messages from the connected check nodes. This summation translates to a convolution of the corresponding pdfs, and the output pdf is given by

$$\mathbf{p}_{v,i}^{[k]} = \mathbf{p}_{\text{AWGN,LLR}} \otimes \left(\mathbf{p}_c^{[k]} \right)^{\otimes(i-1)}. \quad (2.60)$$

Again, due to the random edge interleaving, each edge is connected to a degree i variable node with probability λ_i , such that the pdf $\mathbf{p}_v^{[k]}$ at the input of the check

node decoder is given by

$$\mathbf{p}_v^{[k]} = \mathbf{p}_{\text{AWGN,LLR}} \circledast \sum_{i=2}^{d_v} \lambda_i \cdot \left(\mathbf{p}_c^{[k]} \right)^{\circledast(i-1)} =: \mathbf{p}_{\text{AWGN,LLR}} \circledast \lambda \left(\mathbf{p}_c^{[k]} \right). \quad (2.61)$$

In summary, the DE recursion for the outgoing variable node pdf after the k -th iteration can be written as

$$\mathbf{p}_v^{[k]} = \mathbf{p}_v^{[0]} \circledast \lambda \left(\Gamma^{-1} \left(\rho \left(\Gamma \left(\mathbf{p}_v^{[k-1]} \right) \right) \right) \right), \quad \text{for } k > 0, \quad (2.62)$$

with $\mathbf{p}_v^{[0]} = \mathbf{p}_{\text{AWGN,LLR}}$. Since the all-zero code word was sent, a variable node message is in error if and only if it carries a negative LLR. The probability of an erroneous variable node message after the k -th iteration, thus, results in

$$\mathbf{P}_{\text{error}}^{[k]} = \int_{-\infty}^0 \mathbf{p}_v^{[k]}(x) dx. \quad (2.63)$$

If $\mathbf{P}_{\text{error}}^{[k]} \rightarrow 0$ for $k \rightarrow \infty$, the hard decisions of all messages are 0, resulting in asymptotically error-free decoding, as transmission of the all-zero code word is considered due to the linearity of the LDPC code.

The DE recursion (2.62) only depends on the degree distribution pair (λ, ρ) and the channel quality E_s/N_0 . Thus, to check if a channel quality E_s/N_0 lies above the ensemble threshold $\tau(\lambda, \rho)$, it has to be evaluated whether $\mathbf{P}_{\text{error}}^{[k]} \rightarrow 0$ for $k \rightarrow \infty$; otherwise, the tested channel quality lies below the ensemble threshold. To determine $\tau(\lambda, \rho)$ up to a desired precision, a binary search strategy can be employed. Starting in the middle of a range of channel qualities, the DE recursion is evaluated until either $\mathbf{P}_{\text{error}}^{[k]} \approx 0$ or $\mathbf{P}_{\text{error}}^{[k]}$ stops decreasing, i.e., $\mathbf{P}_{\text{error}}^{[k]} \not\prec \mathbf{P}_{\text{error}}^{[k-1]}$ for a sufficiently large number of iterations. Depending on the outcome, the considered range is either set to the lower or upper half of the previous range. This process is repeated until the range containing the threshold $\tau(\lambda, \rho)$ has reached the target precision.

It should be noted that the evaluation of the DE recursion only describes the asymptotic behavior of an ensemble for $N \rightarrow \infty$. At finite code lengths however, the presence of cycles of length $2l$ violates the assumption of statistical independence of messages after l iterations. Consequently, for all practical code lengths the ensemble threshold can be understood as a lower bound for the channel quality at which near error-free decoding can be achieved. In [CFRU01] a code of length 10^7 was presented that achieves BER 10^{-6} only 0.025 dB above its ensemble threshold. Furthermore, it was shown that for large maximum variable node degrees d_v , the ensemble threshold converges towards the channel capacity [RSU01, CFRU01].

The problem of computing the DE recursion (2.62) remains, since in general it cannot be analytically solved for continuous pdfs. Two approaches to deal with this issue are reviewed in the following: the so-called *Gaussian approximated density evolution* (GADE) [CRU01] and the *discretized density evolution* (DDE) [CFRU01].

Gaussian Approximated Density Evolution (GADE) [CRU01]

In GADE the pdfs of all messages are approximated by Gaussian distributions that fulfill the so-called *symmetry condition* $f(x) = f(-x)e^x$ [RSU01], which for Gaussian distributions $\mathcal{N}(\mu, \sigma^2)$ reduces to $\sigma^2 = 2\mu$. This way, each message pdf can be fully described by only its mean μ , and the DE recursion (2.62) simplifies to an expression that is analytically tractable as follows. The mean of the channel output pdf is denoted μ_L as before. The mean of an outgoing message of a degree- i variable node after the k -th iteration is denoted $\mu_{v,i}^{[k]}$ and that of a degree- j check node is denoted $\mu_{c,j}^{[k]}$, respectively. Taking the mean on both sides of the variable node update equation (2.42) for a degree- i variable node yields the mean

$$\mu_{v,i}^{[k]} = \mu_L + (i-1)\mu_c^{[k]}. \quad (2.64)$$

with $\mu_c^{[k]} = \sum_{j=2}^{d_c} \rho_j \mu_{c,j}^{[k]}$ due to random edge interleaving. Additionally, taking the mean on both sides of the check node update equation (2.41), the following relation can be derived [CRU01]

$$\mu_{c,j}^{[k]} = \phi^{-1} \left(1 - \left(1 - \sum_{i=2}^{d_v} \lambda_i \phi \left(\mu_{v,i}^{[k]} \right) \right)^{j-1} \right), \quad (2.65)$$

where the function $\phi(x)$ for $x \geq 0$ is defined as

$$\phi(x) = \begin{cases} 1 - \frac{1}{\sqrt{4\pi x}} \int_{-\infty}^{\infty} \tanh\left(\frac{u}{2}\right) e^{-\frac{(u-x)^2}{4x}} du, & \text{for } x > 1 \\ 1, & \text{for } x = 0. \end{cases} \quad (2.66)$$

Finally, the GADE recursion is given by

$$\mu_c^{[k]} = \sum_{j=2}^{d_c} \rho_j \phi^{-1} \left(1 - \left(1 - \sum_{i=2}^{d_v} \lambda_i \phi \left(\mu_L + (i-1)\mu_c^{[k-1]} \right) \right)^{j-1} \right), \quad (2.67)$$

for $k \geq 1$ and with $\mu_c^{[0]} = 0$. In this form, the error probability $P_{\text{error}}^{[k]}$ goes to zero if and only if $\mu_c^{[k]} \rightarrow \infty$ for $k \rightarrow \infty$. The lowest channel quality for which this is the case is called the *GADE threshold* $\tau_{\text{GADE}}(\lambda, \rho)$.

The computational complexity of computing the GADE threshold is very low, making it suitable for rapid performance evaluation during ensemble optimization. However, the assumption of pdfs being Gaussian does not exactly model the behavior of the BP decoder, since especially the check node output pdf does not match a Gaussian distribution very well. Thus, the values of GADE thresholds $\tau_{\text{GADE}}(\lambda, \rho)$ only approximate the true ensemble thresholds $\tau(\lambda, \rho)$. As shown further below in Figure 2.4 on page 31 for some exemplary ensembles, the approximation becomes better for higher code rates r and also for lower maximum variable node degrees d_v .

Discretized Density Evolution [CFRU01]

Rather than restricting pdfs to certain forms, the approach of DDE models a message-passing decoder in which all incoming and outgoing LLR messages (including the channel LLRs) are (uniformly) quantized according to

$$\mathcal{Q}(x) = \begin{cases} \left\lceil \frac{\max(x, x_{\min})}{\Delta x} - \frac{1}{2} \right\rceil \cdot \Delta x, & \text{for } x \leq -\frac{\Delta x}{2} \\ 0, & \text{for } -\frac{\Delta x}{2} \leq x \leq \frac{\Delta x}{2} \\ \left\lfloor \frac{\min(x, x_{\max})}{\Delta x} + \frac{1}{2} \right\rfloor \cdot \Delta x, & \text{for } x \geq \frac{\Delta x}{2} \end{cases}, \quad (2.68)$$

where $\Delta x = \frac{x_{\max} - x_{\min}}{2^b - 1}$ is the quantization interval width, b is the number of quantization bits, and $x_{\max}$ and $x_{\min}$ are the maximum and minimum LLR values to be represented. In this quantized form all messages are characterized by a discrete *probability mass function* (pmf) $P(x = c\Delta x) = P(c\Delta x)$, containing the 2^b probabilities for $c = \lceil \frac{x_{\min}}{\Delta x} - \frac{1}{2} \rceil, \dots, -1, 0, 1, \dots, \lfloor \frac{x_{\max}}{\Delta x} + \frac{1}{2} \rfloor$. The variable node update then results in the discrete convolution

$$P_v^{[k]} = P_{\text{AWGN,LLR}} \circledast \sum_{i=2}^{d_v} \lambda_i \left(P_c^{[k]} \right)^{\circledast(i-1)}, \quad (2.69)$$

with $P_v^{[k]}$ and $P_c^{[k]}$ denoting the pmf of incoming check node messages and incoming variable node messages, respectively, both at the k -th iteration, and $P_{\text{AWGN,LLR}}$ denoting the pmf of the quantized channel LLRs. Each of the discrete convolutions of two 2^b -point pmfs in (2.69) can be efficiently computed using a 2^b -point *Fast Fourier Transform* (FFT). The output pmf $P_{c,j}^{[k]}$ of a degree j check node can be computed by

$$P_{c,j}^{[k]}(c\Delta x) = \sum_{\substack{(c_1, \dots, c_{j-1}): \\ \mathcal{Q}(c_1\Delta x \boxplus \dots \boxplus c_{j-1}\Delta x) = c\Delta x}} \prod_{j'=1}^{j-1} P_v^{[k-1]}(c_{j'}\Delta x), \quad (2.70)$$

for $c = \lceil \frac{x_{\min}}{\Delta x} - \frac{1}{2} \rceil, \dots, \lfloor \frac{x_{\max}}{\Delta x} + \frac{1}{2} \rfloor$. Thus, the probability $P_{c,j}^{[k]}(c\Delta x)$ that the outgoing LLR is equal to $c\Delta x$ is computed by summing over all possible combinations of the $j-1$ quantized input LLRs $c_1\Delta x, \dots, c_{j-1}\Delta x$ such that their boxplus sum equals $c\Delta x$ (after quantization). After random edge interleaving the pmf of the variable node input results in

$$P_c^{[k]}(c\Delta x) = \sum_{j=2}^{d_c} \rho_j \sum_{\substack{(c_1, \dots, c_{j-1}): \\ \mathcal{Q}(c_1\Delta x \boxplus \dots \boxplus c_{j-1}\Delta x) = c\Delta x}} \prod_{j'=1}^{j-1} P_v^{[k-1]}(c_{j'}\Delta x). \quad (2.71)$$

The DDE recursion is thus given by the alternating computation of (2.69) and (2.71) for $k \geq 1$, where the initial pmf $P_c^{[0]} = \delta(c\Delta x)$ has all mass at $c\Delta x = 0$. The

probability of a check-to-variable message to be in error after k iterations is computed by

$$P_{c,\text{error}}^{[k]} = \sum_{c=\lceil \frac{x_{\min}}{\Delta x} - \frac{1}{2} \rceil}^{-1} P_c^{[k]}(c\Delta x) + \frac{1}{2} P_c^{[k]}(0), \quad (2.72)$$

i.e., the sum over the probabilities of all negative quantization intervals and half of the quantization interval around zero. The lowest channel quality for which $P_{c,\text{error}}^{[k]} \rightarrow \infty$ for $k \rightarrow \infty$ is denoted the *DDE threshold* $\tau_{\text{DDE}}(\lambda, \rho)$. This threshold approaches the true ensemble threshold $\tau(\lambda, \rho)$ for an increasing number b of quantization bits. However, the computation of the pdf after the check node update (2.70) entails a computational complexity that grows quadratically with the number of quantization intervals, i.e., it lies in $\mathcal{O}(2^{2b})$, which quickly becomes infeasible if a high resolution is used, e.g., in global optimization schemes.

Exemplary Ensemble Thresholds

Several exemplary degree distribution pairs (λ, ρ) of optimized LDPC code ensembles with maximum variable node degrees $d_v \in \{4, 30\}$ and code rates $r \in \{0.1, \dots, 0.9\}$ are given in Table 2.1. All have so-called *concentrated* check node degree distributions $\rho(x) = \rho_j x^{j-1} + \rho_{j+1} x^j$ containing only two consecutive degrees j and $j+1$ [CRU01].

	r	$\lambda(x)$						$\rho(x)$
		λ_2	λ_3	λ_4	λ_5	λ_{20}	λ_{30}	
$d_v = 4$	0.1	0.618	0.086	0.296	-	-	-	$0.223x + 0.777x^2$
	0.2	0.552	0.105	0.343	-	-	-	$0.808x^2 + 0.192x^3$
	0.3	0.472	0.077	0.451	-	-	-	$0.145x^2 + 0.855x^3$
	0.4	0.418	0.093	0.489	-	-	-	$0.347x^3 + 0.653x^4$
	0.5	0.379	0.047	0.574	-	-	-	$0.23x^4 + 0.77x^5$
	0.6	0.356	0.146	0.498	-	-	-	$0.865x^6 + 0.135x^7$
	0.7	0.32	0.105	0.575	-	-	-	$0.144x^8 + 0.856x^9$
	0.8	0.275	0.088	0.637	-	-	-	$0.653x^{14} + 0.347x^{15}$
	0.9	0.217	0.228	0.555	-	-	-	$0.068x^{29} + 0.932x^{30}$
$d_v = 30$	0.1	0.349	0.175	0.092	0.092	0.116	0.176	$0.087x^2 + 0.913x^3$
	0.2	0.267	0.205	0.091	0.081	0.189	0.167	$0.089x^3 + 0.911x^4$
	0.3	0.224	0.185	0.109	0.078	0.162	0.242	$0.838x^5 + 0.162x^6$
	0.4	0.218	0.135	0.099	0.164	0.165	0.219	$0.63x^6 + 0.37x^7$
	0.5	0.182	0.163	0.08	0.116	0.167	0.292	$0.307x^8 + 0.693x^9$
	0.6	0.172	0.154	0.113	0.047	0.326	0.188	$0.327x^{11} + 0.673x^{12}$
	0.7	0.15	0.141	0.106	0.109	0.195	0.299	$0.45x^{16} + 0.55x^{17}$
	0.8	0.127	0.138	0.141	0.071	0.181	0.342	$0.13x^{26} + 0.87x^{27}$
	0.9	0.114	0.141	0.161	0.078	0.171	0.335	$0.349x^{54} + 0.651x^{55}$

Table 2.1: LDPC code ensembles (λ, ρ) with maximum variable node degrees $d_v \in \{4, 30\}$ and code rates $r \in \{0.1, \dots, 0.9\}$.

The thresholds $\tau_{\text{GADE}}(\lambda, \rho)$ and $\tau_{\text{DDE}}(\lambda, \rho)$ of all degree distribution pairs are computed using GADE and DDE. Since both the complexity as well as the accuracy of the threshold prediction of DDE increase with the number b of quantization bits, different values of $b \in \{4, 5, 6, 7, 8, 12\}$ are examined to identify which value offers the best trade-off between complexity and accuracy. The range for the pdfs of DDE was empirically chosen as $[x_{\min}, x_{\max}] = [-5, 30]$.

For an easier comparison of different code rates, the gaps $\gamma(\lambda, \rho)$ in dB of the found thresholds to the Shannon capacity at the corresponding rates are shown in Figure 2.4. In the top left plot, gaps for the $d_v = 4$ ensembles are depicted over the code rate. Since for more than about $b = 8$ quantization bits the computed DDE thresholds are no longer changing, their values can be assumed to closely match the true ensemble thresholds. Both the prediction of thresholds by DDE with $b = 6$ quantization bits and by GADE slightly overestimate the gap to capacity by about 0.1 dB on average,

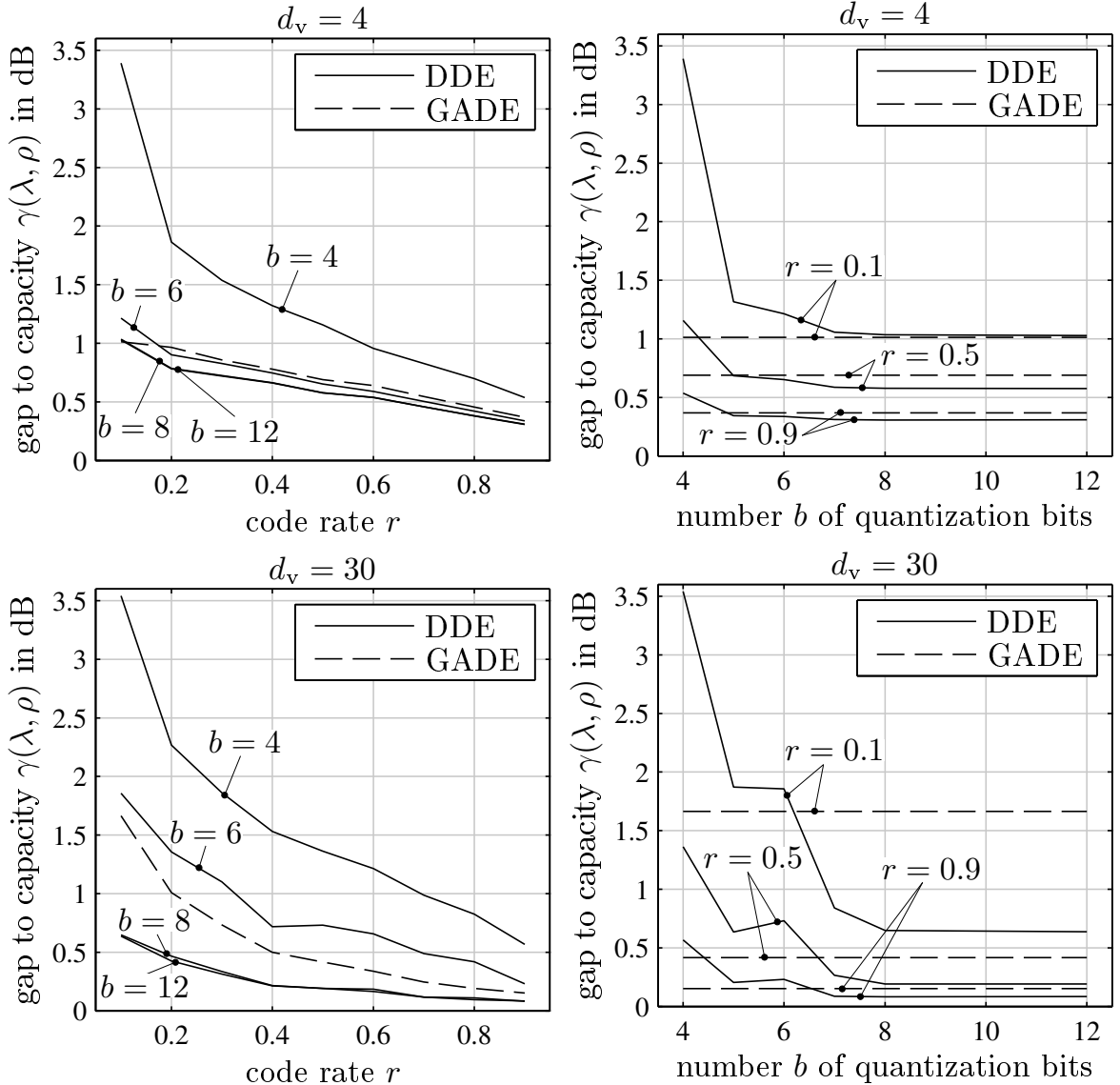


Figure 2.4: Comparison of thresholds of ensembles from Table 2.1, computed with GADE and DDE with different numbers of quantization bits.

where gaps are predicted more precisely with increasing code rate. For DDE with only $b = 4$ bits the estimation error is greater than 2 dB at low rates and about 0.2 dB at high rates. In the bottom left plot, a similar behavior can be observed for the ensembles with $d_v = 30$. Again, $b = 8$ bits seem to suffice to predict the thresholds reasonably well, while for 4 bits large prediction errors occur. However, opposed to the case of $d_v = 4$, GADE as well as DDE with $b = 6$ are not capable of predicting the ensemble thresholds with sufficient precision. In agreement with the results in [CRU01], the descending slope of all curves also indicate that at a constant maximum variable node degree, LDPC code ensembles of high rates are capable of performing closer to the theoretical capacity limit than ensembles of low rates.

The plots on the right-hand side of Figure 2.4 show the gaps to capacity in dB over the number of quantization bits for code rates $r \in \{0.1, 0.5, 0.9\}$. It can be observed that all DDE curves converge to their final values at about $b = 8$ quantization bits. Since no quantization is used in GADE, the GADE thresholds are independent of b and are therefore shown as horizontal lines for comparison.

As a summary, it can be stated that using DDE with $b = 8$ quantization bits results in a very good prediction of ensemble thresholds at all code rates up to a maximum variable node degree of at least $d_v = 30$. For even higher variable node degrees it might become necessary to use a finer quantization. The threshold prediction by GADE can be computed very quickly. However, it should only be used for combinations of code rates and maximum variable node degrees for which it is confirmed to yield an adequate precision.

2.7.2 Optimization Problem

Depending on which of the described methods is used to compute the objective function, i.e., the ensemble threshold $\tau(\lambda, \rho)$, different approaches exist to solve the optimization problem of finding a good degree distribution pair. A common simplification is the consideration of only concentrated check node degree distributions $\rho(x) = \rho_j x^{j-1} + \rho_{j+1} x^j$, such that only the variable node degree distribution $\lambda(x)$ has to be optimized. This simplification is motivated by the observation that virtually all good degree distribution pairs have such concentrated check node degree distributions [CRU01], and the restriction to only these does not involve a noticeable performance loss. Throughout this thesis, concentrated check node degree distributions are used unless stated otherwise. Given the variable node degree distribution $\lambda(x)$ and a target code rate r , the check node degree distribution is then fully determined by

$$\begin{aligned} \rho(x) &= \rho_{\lfloor \bar{d}_c \rfloor} x^{\lfloor \bar{d}_c \rfloor - 1} + \left(1 - \rho_{\lfloor \bar{d}_c \rfloor}\right) x^{\lfloor \bar{d}_c \rfloor} \\ &= \frac{\lfloor \bar{d}_c \rfloor}{\bar{d}_c} (1 - (\bar{d}_c - \lfloor \bar{d}_c \rfloor)) x^{\lfloor \bar{d}_c \rfloor - 1} + \left(1 - \frac{\lfloor \bar{d}_c \rfloor}{\bar{d}_c} (1 - (\bar{d}_c - \lfloor \bar{d}_c \rfloor))\right) x^{\lfloor \bar{d}_c \rfloor} \end{aligned} \quad (2.73)$$

with

$$\bar{d}_c = \frac{\bar{d}_v}{1 - r} = \frac{1}{(1 - r) \sum_{i=2}^{d_v} \frac{\lambda_i}{i}}, \quad (2.74)$$

which can be derived from the two relations

$$\rho_j + \rho_{j+1} = \rho_{\lfloor \bar{d}_c \rfloor} + \rho_{\lfloor \bar{d}_c \rfloor + 1} = 1 \quad (2.75)$$

and

$$\bar{d}_c \stackrel{\text{from (2.23)}}{=} \frac{1}{\sum_{j=2}^{d_c} \frac{\rho_j}{j}} = \frac{1}{\frac{\rho_{\lfloor \bar{d}_c \rfloor}}{\lfloor \bar{d}_c \rfloor} + \frac{\rho_{\lfloor \bar{d}_c \rfloor + 1}}{\lfloor \bar{d}_c \rfloor + 1}}. \quad (2.76)$$

Problem Formulation

For a given target code rate r and a maximum variable node degree d_v , a general formulation of the optimization problem of finding the degree distribution pair (λ, ρ) with concentrated $\rho(x)$ and lowest ensemble threshold $\tau(\lambda, \rho)$ is given by

$$\min \quad \tau(\lambda, \rho) \quad (2.77a)$$

$$\text{subject to} \quad \lambda_i \geq 0, \quad i = 2, \dots, d_v \quad (2.77b)$$

$$\lambda(1) = \sum_{i=2}^{d_v} \lambda_i = 1 \quad (2.77c)$$

$$\rho(x) \text{ and } \bar{d}_c \text{ from (2.73) and (2.74)} \quad (2.77d)$$

$$\lambda_2 \stackrel{(2.50)}{<} \frac{e^{r\tau(\lambda, \rho)}}{\sum_{j=2}^{d_c} \rho_j (j-1)} = \frac{e^{r\tau(\lambda, \rho)}}{\lfloor \bar{d}_c \rfloor \frac{2\bar{d}_c - \lfloor \bar{d}_c \rfloor - 1}{\bar{d}_c}}, \quad (2.77e)$$

where constraints (2.77b) and (2.77c) ensure that the coefficients λ_i are non-negative and sum up to one, whereas (2.77d) assures the check node degree distribution to be concentrated and the obtained degree distribution pair to result in code rate r according to (2.23). The additional constraint (2.77e) is the stability condition in the form (2.50) which upper bounds the fraction of degree two variable nodes as a necessary condition for the ensemble to converge to zero-error probability at channel quality $\tau(\lambda, \rho)$.

In this general form and without further approximations, the optimization problem can only be solved using heuristics and global search strategies like random hill climbing algorithms (e.g., in [RSU01]) or genetic algorithms (e.g., *differential evolution* [SP97]) since the objective function (2.77a) is not known to be convex.

Global Optimization Using DDE and Simulated Annealing

The most generally applicable approach is to use DDE with an appropriate quantization resolution – since it gives reliable threshold predictions – and to resort to a global optimization technique. In this thesis, a heuristic based on a *Simulated Annealing*⁸ (SA) strategy [KGJV83] is introduced.

⁸Simulated Annealing was inspired by the annealing of solids in metallurgy in which materials are heated and slowly cooled down to reach low-energy configurations of their crystal structures.

Initialization The optimization variables λ_i are either initialized randomly to values between zero and one such that $\sum_i \lambda_i = 1$, or a predefined starting position can be used. The initial ensemble threshold $\tau = \tau_{\text{DDE}}(\lambda, \rho)$ is computed using DDE. The current local optimum is denoted $\lambda^* = \lambda$ with the corresponding threshold $\tau^* = \tau_{\text{DDE}}(\lambda, \rho)$. The *temperature* T is initially set to $T = 1$ and controls the convergence of the optimization process.

Rearrangement The so-called rearrangement of the current solution λ with threshold τ applies a small random change to the optimization variables λ_i , taking into account all constraints, yielding a new candidate solution $\tilde{\lambda}$. The corresponding concentrated check node degree distribution $\tilde{\rho}(x)$ for the new candidate variable node degree distribution $\tilde{\lambda}(x)$ is computed using (2.73) and (2.74) and the resulting candidate threshold $\tilde{\tau} = \tau_{\text{DDE}}(\tilde{\lambda}, \tilde{\rho})$ is computed. If the candidate threshold $\tilde{\tau}$ is lower than the current threshold τ , the new candidate is accepted as current solution $\lambda = \tilde{\lambda}$, $\rho = \tilde{\rho}$, and $\tau = \tilde{\tau}$. Furthermore, if the current threshold is the lowest of all thresholds that have been seen so far, the current solution is stored as the new optimum $\lambda^* = \tilde{\lambda}$, $\rho^* = \tilde{\rho}$, and $\tau^* = \tilde{\tau}$. If however, $\tilde{\tau}$ is greater than the current τ , the candidate solution will still be accepted as current solution with probability

$$P(\Delta\tau|_{\text{dB}}, T) = \exp\left(-\frac{\Delta\tau|_{\text{dB}}}{c \cdot T}\right) = \exp\left(-\frac{\tilde{\tau}|_{\text{dB}} - \tau|_{\text{dB}}}{c \cdot T}\right), \quad (2.78)$$

which is high if the degradation $\Delta\tau|_{\text{dB}}$ is small and low if $\Delta\tau|_{\text{dB}}$ is high. Additionally, with decreasing temperature T it becomes less likely to accept a degradation of the current solution. This mechanism of probabilistically accepting degradations avoids getting stuck in high local minima, while preserving the general convergence to low threshold values. The constant value c has to be chosen empirically depending on the range of the temperature as well as the expected range of $\Delta\tau|_{\text{dB}}$. For very high values of c the probability $P(\Delta\tau|_{\text{dB}}, T)$ is mostly close to one, and the SA process almost behaves like a pure random search. For very low values of c , on the other hand, almost no degradations are accepted and the SA process degenerates to a pure *downhill search strategy*, which easily gets stuck in local minima.

Iteration and Cooling Down Depending on the desired optimization time, a certain number of rearrangements are performed at a constant temperature T . After that, the temperature T is decreased according to a predefined *annealing schedule*. Here, an exponentially decreasing temperature is employed, such that a relatively large amount of time is spent at low temperatures allowing the optimization to locally converge towards the end of the process.

Freezing the System Several stopping rules can be employed to finalize the optimization process. Commonly, optimization is either stopped if a minimum temperature has been reached or no new solution has been accepted for a predefined number of temperature steps. In both cases the system is considered *frozen*. The final output of the optimizer is the best found degree distribution λ^* with threshold τ^* .

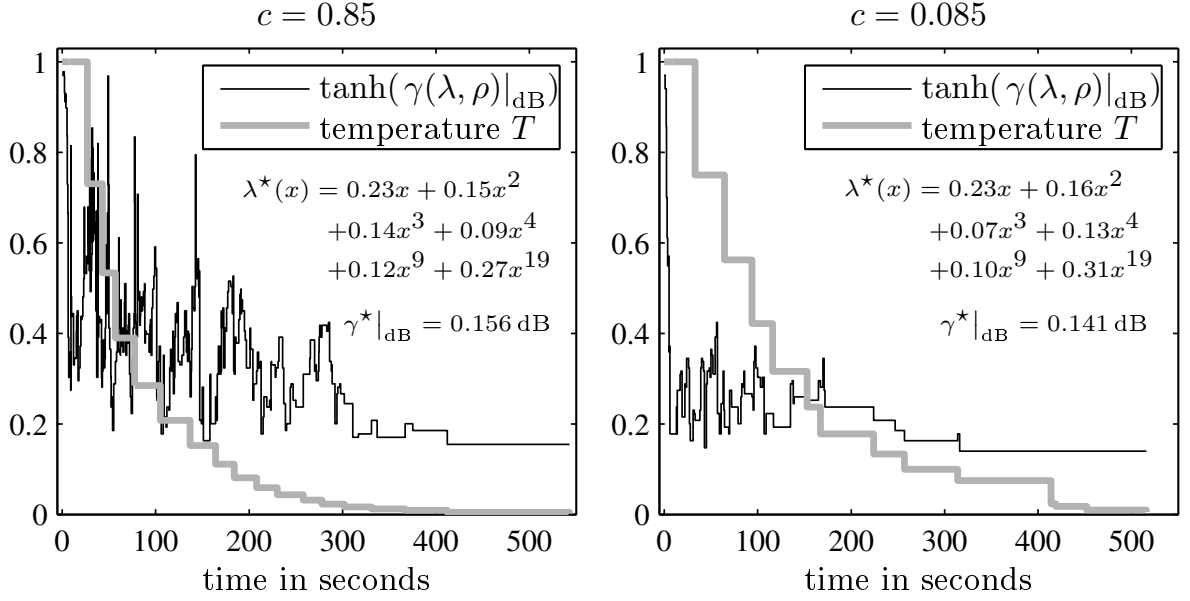


Figure 2.5: Simulated Annealing optimization of a variable node degree distribution with $d_v = 20$ and $r = \frac{1}{2}$. DDE with $b = 8$ quantization bits is used. The bold gray lines show the temperature T and the black lines show the $\tanh(\cdot)$ of the current gap to capacity $\gamma(\lambda, \rho)|_{\text{dB}}$ during optimization. On the left $c = 0.85$ and on the right $c = 0.085$ is chosen.

In Figure 2.5 two exemplary runs of the SA degree distribution optimizer lasting for about 9 minutes on a standard *Intel Core i7* desktop CPU are shown. The temperature T is shown in bold gray lines and the $\tanh(\cdot)$ of the current gap to capacity $\gamma(\lambda, \rho)|_{\text{dB}}$ in dB during the optimization is shown in black.⁹ The target code rate is chosen as $r = \frac{1}{2}$ and the maximum variable node degree as $d_v = 20$. DDE with $b = 8$ quantization bits is used to compute the ensemble thresholds. The annealing schedule is set to exponentially decrease the temperature step-by-step from $T = 1$ to $T = 0.001$. On the left-hand side the constant in (2.78) is set to $c = 0.85$ resulting in a relatively high acceptance probability. For the right-hand side $c = 0.085$ – and thus a lower acceptance probability – is used and a better final solution is found.

Linearization Using GADE

An alternative method which enables a much faster – though less accurate – optimization can be derived by linearizing the optimization problem (2.77) using the Gaussian Approximation (GA) algorithm from above (Section 2.7.1). This linearization was proposed in [CRU01] and is briefly reviewed here. As mentioned before, concentrated check node degree distributions are considered in this thesis.

⁹For an easier illustration and comparison, rather than directly plotting the threshold τ , the $\tanh(\gamma|_{\text{dB}})$ of the corresponding gap to capacity γ in dB is depicted. By using the gap to capacity, the range of values is shifted by a constant offset from $(q(r), +\infty)$ to $(0, +\infty)$. Additionally, taking the $\tanh(\cdot)$, monotonically maps the range $(0, +\infty)$ to $(0, 1)$. Since $\tanh(x) \approx x$ for small values of x , the most relevant scope of thresholds close to capacity is not noticeably biased. This way, the values are in the same range for all setups making them easily comparable.

In the GADE recursion (2.67), the auxiliary variable $t^{[k-1]}$ is introduced by

$$\mu_c^{[k]} \stackrel{(2.67)}{=} \sum_{j=2}^{d_c} \rho_j \phi^{-1} \left(1 - \underbrace{\left(1 - \sum_{i=2}^{d_v} \lambda_i \phi \left(\mu_L + (i-1) \mu_c^{[k-1]} \right) \right)^{j-1}}_{=: t^{[k-1]}} \right)$$

such that the recursion can be rephrased to a variable node perspective as

$$t^{[k]} = \sum_{i=2}^{d_v} \lambda_i \phi \left(\mu_L + (i-1) \underbrace{\sum_{j=2}^{d_c} \rho_j \phi^{-1} \left(1 - \left(1 - t^{[k-1]} \right)^{j-1} \right)}_{=: \mu_c^{[k]}} \right) \quad (2.79)$$

$$=: h(\mu_L, t^{[k-1]}) \quad (2.80)$$

for $k \geq 1$ and with $t^{[0]} = \phi(\mu_L)$. Using this definition of the auxiliary function $h(\mu_L, t)$, it was shown in [CRU01] that the error probability goes to zero if and only if $t^{[k-1]} > t^{[k]}$ for all $k > 0$, which is equivalent to

$$x > h(\mu_L, x), \quad \forall x \in (0, \phi(\mu_L)). \quad (2.81)$$

This inequality is called the *convergence condition* and is a linear expression in all values λ_i , since $h(s, t)$ is linear in λ_i . With the help of this convergence condition, the following optimization strategy can be realized to find the best variable node degree distribution of a given target code rate r_{target} and a given maximum degree d_v . A visualization of the process is presented in Figure 2.6.

At first, a low fixed mean check node degree $\bar{d}_c$, defining the check node degree distribution $\rho(x)$ by (2.73), and a high fixed channel quality $\frac{E_s}{N_0} = \frac{\mu_L}{4}$ are chosen.

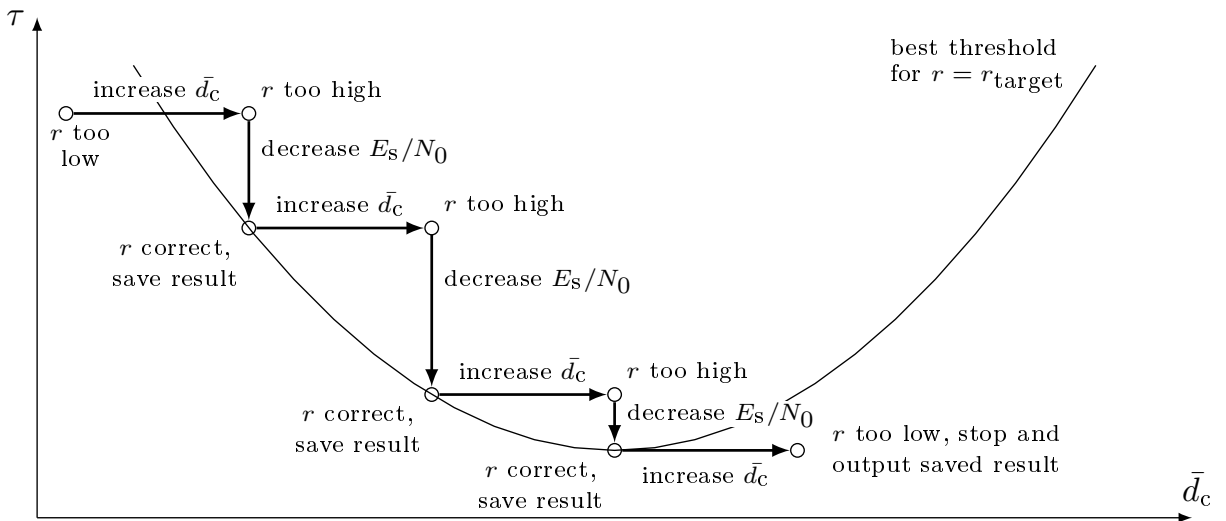


Figure 2.6: Visualization of linearized optimization process by maximizing the code rate. At each circle the linear program (2.82) is solved once. The y -axis represents the current ensemble threshold τ . The x -axis represents the current mean check node degree $\bar{d}_c$.

Then, a variable node degree distribution $\lambda(x)$ is optimized by maximizing the code rate $r \stackrel{(2.23)}{=} 1 - \frac{\sum_j \rho_j/j}{\sum_i \lambda_i/i}$, such that the convergence condition (2.81) as well as the stability condition (2.50) are fulfilled. Since the ρ_j are considered fixed, maximizing r is equivalent to maximizing $\sum_i \lambda_i/i$. This problem of finding the degree distribution with the highest rate is stated in the following *linear program*:

$$\max \quad \sum_{i=2}^{d_v} \frac{\lambda_i}{i} \quad (2.82a)$$

$$\text{subject to} \quad \lambda_i \geq 0, \quad i = 2, \dots, d_v \quad (2.82b)$$

$$\lambda(1) = \sum_{i=2}^{d_v} \lambda_i = 1 \quad (2.82c)$$

$$\lambda_2 < \frac{e^{\frac{r \cdot \mu_L}{4}}}{\sum_{j=2}^{d_c} \rho_j (j-1)} \quad (2.82d)$$

$$x \stackrel{(2.81)}{>} h(\mu_L, x), \quad \forall x \in (0, \phi(\mu_L)), \quad (2.82e)$$

where the objective function as well as all constraints are linear in λ_i . Linear programs can be efficiently solved using, e.g., the *simplex method* or an *interior point method* [BV04]. If the resulting code rate is lower than the target code rate r_{target} , the current channel quality cannot be achieved as threshold with the current check node degree distribution. Thus, the mean check node degree $\bar{d}_c$ is increased slightly, yielding a new check node degree distribution (2.73). Again, the optimization problem (2.82) is solved. This is repeated until the optimized code rate lies above the target code rate. At this point, the channel quality under test is slightly decreased until the output code rate of the optimizer exactly matches the target code rate (up to a desired precision). Once this lowest achievable channel quality is found, the mean check node degree is again slightly increased and the whole process is executed again to check if an even lower channel quality can be achieved with the modified check node degree distribution. Finally, as soon as the channel quality cannot be improved any further by increasing the mean check node degree, the optimization process is stopped and the best degree distribution pair with the desired target code rate has been found.

Linearization Using EXIT Charts

Another well-known approach of transforming the ensemble optimization problem into an efficiently solvable program is to employ *extrinsic information transfer* (EXIT) *charts*, which were originally introduced in [tB01] in the context of Turbo codes and later also applied to LDPC codes, e.g., in [tBKA04, SAL06]. Similarly to the case of Gaussian approximated density evolution, for the application of EXIT charts, the message probability densities are assumed to be Gaussian. Under this assumption, EXIT charts provide a very illustrative description of the iterative decoding process by quantifying the *mutual information* that is exchanged between check nodes and

variable nodes [tBKA04]. Using this description, the ensemble optimization can be transformed into a simple *least squares* curve fitting problem [TH02]. Since, however, EXIT chart based techniques suffer from the same accuracy problems as other methods that use a Gaussian assumption, they are not further elaborated here. Throughout this thesis, all ensemble optimization procedures are instead based on density evolution techniques.

2.8 Code Construction

After having found an optimized degree distribution pair (λ, ρ) or equivalently (Λ, R) in node perspective, the second part in the design of an LDPC code is the construction of a parity check matrix $\mathbf{H}$ of the required size $M \times N$ that complies with the given degree distribution pair. This check matrix construction corresponds to choosing a concrete code realization from the given ensemble (N, λ, ρ) . The first step consists of determining the variable node and check node degree sequences $\mathbf{d}_v$ and $\mathbf{d}_c$ from (2.19), containing the degrees $d_{v,n}$ and $d_{c,m}$ of the n -th variable node and the m -th check node, respectively. The total number of degree- i variable nodes and degree- j check nodes are given by

$$N_i = \Lambda_i N, \quad i = 2, \dots, d_v \quad (2.83)$$

and

$$M_j = R_j M, \quad j = 2, \dots, d_c, \quad (2.84)$$

respectively, where appropriate rounding $\lceil \cdot \rceil$ needs to ensure that $\lceil \Lambda_i N \rceil$ and $\lceil R_j M \rceil$ are mapped to integer values and $\sum_{i=2}^{d_v} \lceil \Lambda_i N \rceil = N$ and $\sum_{j=2}^{d_c} \lceil R_j M \rceil = M$ hold. The variable node degree sequence in ascending order is, thus, given by

$$\mathbf{d}_v = (d_{v,1}, \dots, d_{v,N}) = (\underbrace{2, \dots, 2}_{N_2 \text{ times}}, \dots, \underbrace{i, \dots, i}_{N_i \text{ times}}, \dots, \underbrace{d_v, \dots, d_v}_{N_{d_v} \text{ times}}) \quad (2.85)$$

and likewise the check node degree sequence in ascending order by

$$\mathbf{d}_c = (d_{c,1}, \dots, d_{c,M}) = (\underbrace{2, \dots, 2}_{M_2 \text{ times}}, \dots, \underbrace{j, \dots, j}_{M_j \text{ times}}, \dots, \underbrace{d_c, \dots, d_c}_{M_{d_c} \text{ times}}). \quad (2.86)$$

Each variable (check) node n (m) can now be seen as having $d_{v,n}$ ($d_{c,m}$) free *sockets* that need to be connected to different check (variable) nodes. Choosing a random code from the ensemble, thus, means to find a random permutation from the $\sum_{n=1}^N d_{v,n} = E$ variable node sockets to the $\sum_{m=1}^M d_{c,m} = E$ check node sockets, where E denotes the total number of edges. Thereby, *double edges* have to be avoided, i.e., no two sockets of one node may be connected to two sockets of another node. Commonly, this random construction is extended by the additional rule of not connecting any two variable nodes to two (or more) common check nodes, which avoids the creation of cycles of length 4 in the Tanner graph.

Besides the described random construction, two widely-used methods for constructing an LDPC check matrix from a given degree distribution pair are the *progressive edge-growth* (PEG) algorithm [HEA01, HEA05] and the ACE constrained construction presented in [TJVW04]. In the following, the original PEG algorithm and the *improved PEG* algorithm [XB04], which incorporates the ACE optimization proposed in [TJVW04], are briefly reviewed, as they are later used as a starting point for an optimized multi-rate code construction method developed in Chapter 3.

2.8.1 Progressive Edge-Growth [HEA05]

The PEG algorithm was first presented in [HEA01, HEA05]. It is a *greedy algorithm* that constructs the Tanner graph of an LDPC check matrix of given size $M \times N$ and variable node degree sequence $\mathbf{d}_v$ with ascending degrees in a variable-node-by-variable-node manner trying to maximize its girth g . Starting at the lowest index, each variable node n is connected to $d_{v,n}$ check nodes one after another, where the objective in each step is to maximize the length of the cycle that is (potentially) created by the edge to be established next. For this purpose, in each step, starting from the current variable node n , the tree $\mathcal{T}_n^l$ is constructed (see (2.28)) up to the depth l at which either no new nodes can be added from depth l to depth $l + 1$ or all nodes are included in $\mathcal{T}_n^l$. In the first case, an edge is added between the current variable node and a random check node of minimal current degree amongst all check nodes that are currently not reachable, since this edge will not create a new cycle. In the latter case, all check nodes can already be reached and the edge is connected to a random check node of minimal current degree from the neighborhood $\mathcal{M}^l(n)$ of depth l (see (2.27)). In this case, the creation of a new cycle cannot be avoided, but its length $l + 1$ is maximized by selecting one of the most distant check nodes. The PEG construction is summarized in Algorithm 1.

No explicit check node degree distribution is given as input to the PEG algorithm. Instead, since connected check nodes are always chosen from all candidates with currently minimal degree, the degrees of all check nodes grow uniformly, resulting in (nearly) concentrated check node degree distributions. However, if a specific check node degree distribution is desired, a straightforward modification of the PEG algorithm can be used, in which each check node is assigned a number of sockets according to an targeted check node degree sequence $\mathbf{d}_c$. Once all sockets of a check node are connected, it is no longer available as a candidate during further PEG processing [Win11].

2.8.2 Improved Progressive Edge-Growth [XB04]

A specific extension of the PEG algorithm to improve the ACE properties of the resulting codes was proposed in [XB04] and is simply termed *improved PEG* (iPEG). It introduces a modified rule of choosing a check node to connect to, only for the case that new cycles cannot be avoided (lines 10 – 11 in Algorithm 1) and multiple candidates with the same degree are present at maximum depth. Where the original

Algorithm 1 Progressive Edge-Growth (PEG) algorithm [HEA05].

```

1: Input:  $M, N$ , and  $\mathbf{d}_v = (d_{v,1}, \dots, d_{v,N})$ 
2: Initialize  $\mathcal{V} = \{1, \dots, N\}$ ,  $\mathcal{C} = \{1, \dots, M\}$ , and  $\mathcal{E} = \emptyset$ 
3: Initialize  $\mathbf{d}_c = (d_{c,1}, \dots, d_{c,M}) = (0, \dots, 0)$ 
4: for  $n = 1, \dots, N$  do
5:   Connect first edge to random check node of minimal degree:
6:    $\mathcal{E} = \mathcal{E} \cup \{(n, m)\}$  and  $d_{c,m} = d_{c,m} + 1$ , with  $m \in \arg \min_{m' \in \mathcal{C}} d_{c,m'}$ 
7:   for  $i = 2, \dots, d_{v,n}$  do
8:     Expand  $\mathcal{T}_n^l$  up to depth  $l$  until no nodes are added from depth  $l$  to  $l + 1$ 
9:     if  $\mathcal{T}_n^l$  contains all check nodes then
10:      Connect edge to random check node at maximal depth of minimal degree:
11:       $\mathcal{E} = \mathcal{E} \cup \{(n, m)\}$  and  $d_{c,m} = d_{c,m} + 1$ , with  $m \in \arg \min_{m' \in \mathcal{M}^l(n)} d_{c,m'}$ 
12:    else
13:      Connect edge to random unreachable check node of minimal degree:
14:       $\mathcal{E} = \mathcal{E} \cup \{(n, m)\}$  and  $d_{c,m} = d_{c,m} + 1$ , with  $m \in \arg \min_{m' \notin \mathcal{T}_n^l} d_{c,m'}$ 
15:    end if
16:  end for
17: end for

```

PEG algorithm would simply randomly choose a check node in this case, the iPEG algorithm chooses the check node that maximizes the minimum ACE value that is created by the new cycles. This modified rule does not influence the length of the newly introduced cycles in each step but avoids the creation of cycles with low ACE value whenever possible. This method was shown to improve the error correction performance especially in the error floor region since cycles with low ACE values are known to have particularly detrimental effects.

2.9 Quasi-Cyclic LDPC Codes

A drawback of the code construction methods presented above is that the resulting check matrices do not possess any special structure, i.e., their entries are arranged in a random manner. This makes storing the check matrix inefficient and is also unfavorable for parallelizing the decoder due to the random data dependencies during the variable and check node updates. For these reasons, most LDPC codes that are implemented today are so-called *quasi-cyclic* (QC) LDPC codes [KLF01]. They have a parity check matrix with a structure that allows for inherent parallelization and that can be efficiently stored. Furthermore, very efficient methods for fast encoding of QC LDPC codes exist [MYK05, LCZ⁺06]. Almost all LDPC codes that are utilized in practice belong to the class of QC LDPC codes. The following description of QC LDPC codes is given for the binary case only, but can also be extended to the non-binary case [PC07, HLZZ10, BPL13].

The $M \times N$ check matrix of a QC LDPC code is given by

$$\mathbf{H}^{[Z]} = \begin{pmatrix} \mathbf{I}_s^{A_{1,1}} & \mathbf{I}_s^{A_{1,2}} & \cdots & \mathbf{I}_s^{A_{1,N'}} \\ \mathbf{I}_s^{A_{2,1}} & \mathbf{I}_s^{A_{2,2}} & \cdots & \mathbf{I}_s^{A_{2,N'}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{I}_s^{A_{M',1}} & \mathbf{I}_s^{A_{M',2}} & \cdots & \mathbf{I}_s^{A_{M',N'}} \end{pmatrix}, \quad (2.87)$$

with the integer *lifting factor* $Z \in \mathbb{N}^+$ and $N' = N/Z$ as well as $M' = M/Z$. The submatrices $\mathbf{I}_s^{A_{m',n'}}$ in (2.87) are either all-zero matrices of size $Z \times Z$ or identity matrices of size $Z \times Z$ cyclically shifted to the right by $A_{m',n'}$ positions.¹⁰ Thus, $\mathbf{I}_s^{A_{m',n'}}$ can be represented as

$$\mathbf{I}_s^{A_{m',n'}} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix}^{A_{m',n'}}$$

where $\mathbf{I}_s$ denotes the $Z \times Z$ identity matrix cyclically shifted to the right by one position. Each multiplication of $\mathbf{I}_s$ with itself leads to an additional cyclic right shift by one position. Slightly abusing notation, the matrix $\mathbf{I}_s^{-1}$ is defined as the $Z \times Z$ all-zero matrix. The QC check matrix can thus be fully described by only the shift values $A_{m',n'}$ which are assembled in the so-called *lifting matrix* of dimension $M' \times N'$:

$$\mathbf{A} = \begin{pmatrix} A_{1,1} & \cdots & A_{1,N'} \\ \vdots & \ddots & \vdots \\ A_{M',1} & \cdots & A_{M',N'} \end{pmatrix}. \quad (2.88)$$

Groups of Z consecutive columns or rows (variable or check nodes) in the *expanded* check matrix $\mathbf{H}^{[Z]}$ corresponding to one entry $A_{m',n'}$ in $\mathbf{A}$ (i.e., columns in the range $(n' - 1)Z + 1, \dots, n'Z$ and rows in the range $(m' - 1)Z + 1, \dots, m'Z$) are referred to as a *block* or *group* of columns or rows (variable or check nodes), respectively. An important property of QC LDPC codes is that no variable node is connected to one such block of check nodes more than once. This property is important for realizing efficient parallel decoders and is exploited by the *layered BP* implementation presented in Section 6.3.

Due to the unity column and row weights of the (shifted) identity matrix $\mathbf{I}_s^{A_{m',n'}}$, both column and row degree distributions of the expanded parity check matrix $\mathbf{H}^{[Z]}$

¹⁰Quasi-cyclic LDPC codes can also be more generally defined as being composed of arbitrary square circulant matrices, in which each row is the cyclic right shift by one position to the right of the row above. By most authors, however, these circulants are restricted to be cyclically shifted identity matrices, which enables many advantages regarding the hardware implementation. As long as the dimensions of the lifting matrix $\mathbf{A}$ are sufficiently large, this restriction does not impose a notable loss in error correction performance.

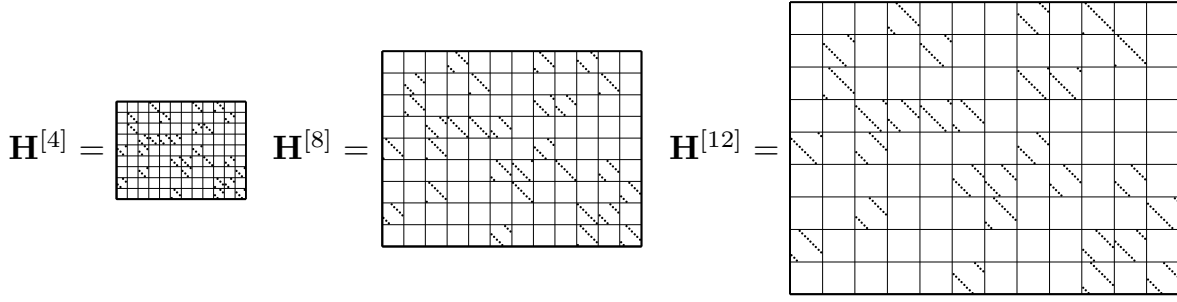


Figure 2.7: Three exemplary QC check matrices constructed by replacing each 1-entry in the example matrix (2.31) by a randomly shifted identity matrix of size $Z \in \{4, 8, 12\}$. All three matrices have the same code rate and degree distribution pair as the matrix in (2.31), but different code lengths $N'Z = N \in \{48, 96, 144\}$. Each dot represents a 1-entry and the grid lines illustrate the $Z \times Z$ block structure.

are equivalent to the column and row degree distributions of the lifting matrix $\mathbf{A}$ if the weight of a column (row) in $\mathbf{A}$ is defined as the number of entries $A_{m',n'} \neq -1$. Therefore, the design of a QC LDPC code ensemble can be reduced to optimizing the degree distribution pair of the lifting matrix, as all check matrices expanded from this lifting matrix with arbitrary lifting factor Z will maintain the same degree distribution pair. For the construction of a concrete QC code realization the positions of the non-zero submatrices as well as the corresponding shift values have to be determined. For this, variations of the above presented construction methods as PEG or ACE can be employed by adding an additional *circulant constraint* to the selection of matrix entries which ensures that a check matrix is constructed consisting of either all-zero matrices or cyclically shifted identity matrices each of size $Z \times Z$ (cf. [LK04]). Some exemplary randomly constructed QC check matrices are depicted in Figure 2.7.

In the context of QC LDPC codes, the concept of a so-called *protograph* description is commonly used. The protograph representation of a lifting matrix $\mathbf{A}$ is a (small) Tanner graph with N' variable nodes, M' check nodes, and connections between variable and check nodes according to the positions of non-negative entries in $\mathbf{A}$ (where the exact value is irrelevant). Each node of this protograph defines a certain type of node in terms of how many connections it has and to which other types of nodes it is connected. In the example in Figure 2.8, e.g., each check node of type “2” has to be connected to exactly one variable node of type “a” and one variable node of type “c”.

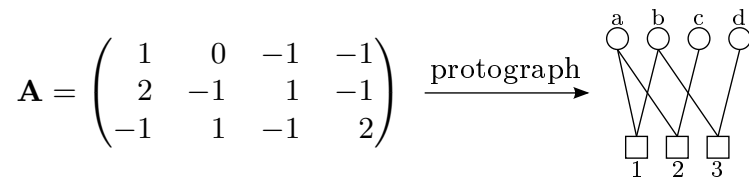


Figure 2.8: Exemplary 3×4 lifting matrix $\mathbf{A}$ and its protograph representation with 4 variable nodes and 3 check nodes.

2.10 Non-Binary LDPC Codes

It has been shown that the error correction performance of LDPC codes under message-passing decoding can be improved by extending codes to higher order finite fields $\mathbb{F}_q$ yielding so called non-binary LDPC codes [DM98a, DM98b]. Especially for short to medium code lengths these non-binary LDPC codes are promising candidates for achieving further gains compared to traditional binary LDPC codes, however, at the cost of an increased decoding complexity. The only difference in the definition of binary and non-binary LDPC codes is that in case of non-binary LDPC codes the information words $\mathbf{x} \in \mathbb{F}_q^{Kq}$, the code words $\mathbf{y} \in \mathbb{F}_q^{Nq}$, and the check matrix $\mathbf{H} \in \mathbb{F}_q^{Mq \times Nq}$ are defined over the finite field $\mathbb{F}_q$, with $q = 2^p$ and $p > 1$.

2.10.1 Transmission System and Non-Binary LLR Vectors

To comply with the transmission system model introduced in Section 2.1, a slightly modified notation is used. The binary information and code words are denoted $\mathbf{x}^{\text{bin}}$ and $\mathbf{y}^{\text{bin}}$, respectively. The transmission system still operates in the binary representation as introduced above, only differing in the internal representation of values inside the LDPC encoder and decoder, as illustrated in Figure 2.9.

The binary input word $\mathbf{x}^{\text{bin}} \in \mathbb{F}_2^K$ of length K is converted to a q -ary input word $\mathbf{x} \in \mathbb{F}_q^{Kq}$ of length $Kq = \frac{K}{p}$ by grouping p successive bits from $\mathbf{x}^{\text{bin}}$ at a time and mapping them to the field element $a \in \mathbb{F}_q$ with the corresponding binary representation¹¹. Encoding is then performed as described in Section 2.4 using the q -ary check matrix $\mathbf{H} \in \mathbb{F}_q^{Mq \times Nq}$ of code rate $r = \frac{Kq}{Nq}$. The resulting q -ary code word $\mathbf{y} \in \mathbb{F}_q^{Nq}$ of length $Nq = \frac{N}{p}$ consists of Kq information symbols and $Mq = \frac{M}{p}$ parity symbols.

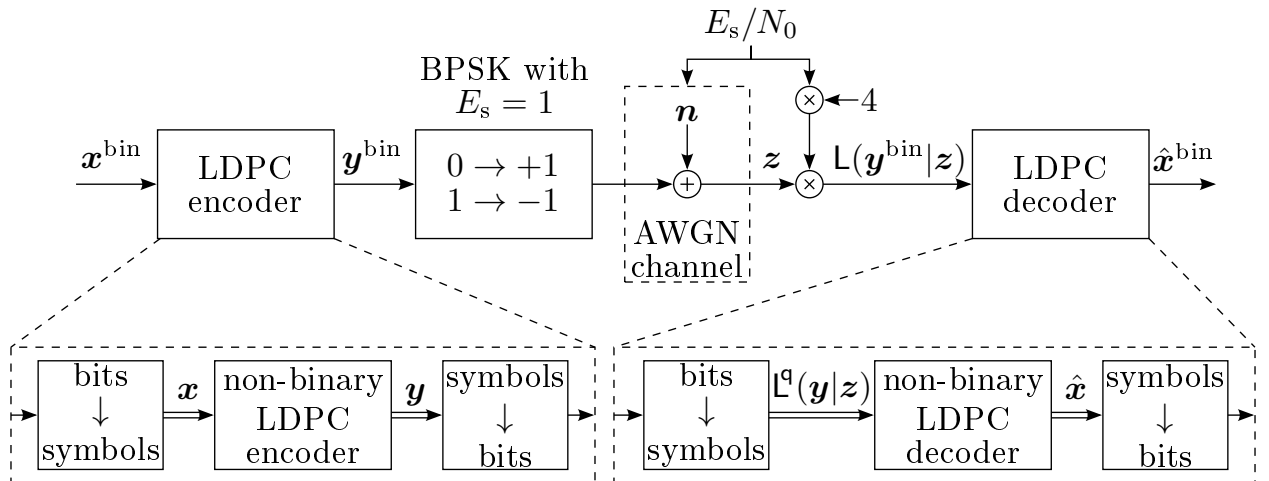


Figure 2.9: Transmission system with binary input and output data, a binary input AWGN channel, and non-binary LDPC encoding and decoding.

¹¹A detailed description of how finite fields are generated and represented can be found, e.g., in [RL09]

Each code symbol is then mapped back to p code bits by expanding the field element to its binary representation, yielding the binary code word $\mathbf{y}^{\text{bin}} \in \mathbb{F}_2^N$. This binary code word can then be mapped to BPSK symbols and transmitted over the binary input AWGN channel, after which the receiver computes binary channel LLRs¹² $L(\mathbf{y}^{\text{bin}}|\mathbf{z})$ from the channel output $\mathbf{z}$ just like in the pure binary case. While each of these binary LLRs describes the (conditional) probabilities of one code bit being equal to 0 or 1, for the non-binary LDPC decoder a suitable corresponding logarithmic domain representation of the q probabilities of a code symbol being equal to each of the q field elements (conditioned on the respective channel output values) is required. For this purpose, the notion of a q -dimensional (conditional) LLR vector [WSM04] is introduced

$$\begin{aligned} \mathbf{L}^q(y_n | z_{(n-1)p+1}, \dots, z_{np}) &= \left(\log \frac{P(y_n = 0 | z_{(n-1)p+1}, \dots, z_{np})}{P(y_n = \alpha^{q-2} | z_{(n-1)p+1}, \dots, z_{np})}, \right. \\ &\quad \left. \log \frac{P(y_n = 1 | z_{(n-1)p+1}, \dots, z_{np})}{P(y_n = 0 | z_{(n-1)p+1}, \dots, z_{np})}, \dots, \log \frac{P(y_n = \alpha^{q-2} | z_{(n-1)p+1}, \dots, z_{np})}{P(y_n = 0 | z_{(n-1)p+1}, \dots, z_{np})} \right) \\ &=: (L_n^{[0]}, L_n^{[1]}, \dots, L_n^{[\alpha^{q-2}]}) = \mathbf{L}_n. \end{aligned} \quad (2.89)$$

The conditioning of y_n on the p channel output values $z_{(n-1)p+1}, \dots, z_{np}$ is due to the fact that these values are the noisy observations of the p successive code bits $y_{(n-1)p+1}^{\text{bin}}, \dots, y_{np}^{\text{bin}}$ from $\mathbf{y}^{\text{bin}}$ from which the code symbol y_n has been assembled. Although the first entry seems redundant as it is always equal to zero, it is often still included for implementational reasons.¹³ Let $a \in \mathbb{F}_q$ be an arbitrary field element with binary representation $a = a_1 a_2 \dots a_p$, with $a_i \in \mathbb{F}_2$. Under the assumption of independent channel noise samples, the conditional probability of a code symbol y_n being equal to a can then be rewritten as

$$\begin{aligned} &P\left(\underbrace{y_n = a}_{\downarrow} | z_{(n-1)p+1}, \dots, z_{np}\right) \\ &= P\left(\overbrace{y_{(n-1)p+1}^{\text{bin}} = a_1, \dots, y_{np}^{\text{bin}} = a_p} | z_{(n-1)p+1}, \dots, z_{np}\right) \\ &= P\left(y_{(n-1)p+1}^{\text{bin}} = a_1 | z_{(n-1)p+1}\right) \cdot \dots \cdot P\left(y_{np}^{\text{bin}} = a_p | z_{np}\right) \\ &= \prod_{i=1}^p P\left(y_{(n-1)p+i}^{\text{bin}} = a_i | z_{(n-1)p+i}\right). \end{aligned} \quad (2.90)$$

Using (2.90) and the fact that the binary representation of $0 \in \mathbb{F}_q$ is $\underbrace{0 \dots 0}_{p \text{ times}}$, the LLR

¹²It should be noted that, in this context, the term binary LLR does not implicate the LLR itself having a binary value, but that the LLR describes the probabilities of a binary random variable.

¹³Of course, just like in the binary case, only $q-1$ logarithmic values are required to represent the q probabilities, due to the additional constraint of their sum being equal to one. However, the choice of which field element is used in the denominator of the probability ratios in (2.89) is somewhat arbitrary and might be changed during the decoding process to increase numerical stability.

representation of each field element $a \in \mathbb{F}_q$ can be rephrased to

$$\begin{aligned}
L_n^{[a]} &= \log \frac{\mathbb{P}(y_n = a | z_{(n-1)p+1}, \dots, z_{np})}{\mathbb{P}(y_n = 0 | z_{(n-1)p+1}, \dots, z_{np})} \\
&= \log \frac{\prod_{i=1}^p \mathbb{P}(y_{(n-1)p+i}^{\text{bin}} = a_i | z_{(n-1)p+i})}{\prod_{i=1}^p \mathbb{P}(y_{(n-1)p+i}^{\text{bin}} = 0 | z_{(n-1)p+i})} \\
&= \log \prod_{1 \leq i \leq p: a_i=1} \frac{\mathbb{P}(y_{(n-1)p+i}^{\text{bin}} = 1 | z_{(n-1)p+i})}{\mathbb{P}(y_{(n-1)p+i}^{\text{bin}} = 0 | z_{(n-1)p+i})} \\
&= - \sum_{1 \leq i \leq p: a_i=1} \log \frac{\mathbb{P}(y_{(n-1)p+i}^{\text{bin}} = 0 | z_{(n-1)p+i})}{\mathbb{P}(y_{(n-1)p+i}^{\text{bin}} = 1 | z_{(n-1)p+i})} \\
&= - \sum_{1 \leq i \leq p: a_i=1} \mathbb{L}(y_{(n-1)p+i}^{\text{bin}} | z_{(n-1)p+i}), \tag{2.91}
\end{aligned}$$

where in the summation only those indices are considered at which the binary representation of a has a one entry, since all terms corresponding to zero entries are present in the numerator and denominator and thus cancel out. Using (2.91), each group of p successive binary channel LLRs $\mathbb{L}(y_{(n-1)p+i}^{\text{bin}} | z_{(n-1)p+i}), \dots, \mathbb{L}(y_{np}^{\text{bin}} | z_{np})$ are mapped to one q -dimensional LLR vector $\mathbf{L}_n = \mathbb{L}^q(y_n | z_{(n-1)p+1}, \dots, z_{np})$. These LLR vectors are fed into the non-binary LDPC decoder to produce an estimate $\hat{\mathbf{x}}$ of the q -ary information word $\mathbf{x}$, which is finally mapped to the binary estimate $\hat{\mathbf{x}}^{\text{bin}}$. The relationships between code bits, code symbols, binary LLRs, and non-binary LLR vectors are visualized in Table 2.2.

y_1			...	y_{N_q}		
y_1^{bin}	...	y_p^{bin}	...	$y_{(N_q-1)p+1}^{\text{bin}}$	...	$y_{N_q p}^{\text{bin}}$
$\downarrow$	$\vdots$	$\downarrow$	$\vdots$	$\downarrow$	$\vdots$	$\downarrow$
z_1	$\vdots$	z_p	$\vdots$	$z_{(N_q-1)p+1}$	$\vdots$	$z_{N_q p}$
$\downarrow$	...	$\downarrow$	...	$\downarrow$	...	$\downarrow$
$\mathbb{L}(y_1^{\text{bin}} z_1)$	...	$\mathbb{L}(y_p^{\text{bin}} z_p)$	...	$\mathbb{L}(y_{(N_q-1)p+1}^{\text{bin}} z_{(N_q-1)p+1})$	...	$\mathbb{L}(y_{N_q p}^{\text{bin}} z_{N_q p})$
$\mathbb{L}^q(y_1 z_1, \dots, z_p)$			...	$\mathbb{L}^q(y_{N_q} z_{(N_q-1)p+1}, \dots, z_{N_q p})$		

Table 2.2: Mapping of symbols to bits and binary LLRs to non-binary LLR vectors.

2.10.2 Non-Binary Belief Propagation Decoding¹⁴

A decoding algorithm for non-binary LDPC codes over $\mathbb{F}_q$ has already been described in [DM98a]. This algorithm uses the *Fast Fourier Transform* (FFT) over $\mathbb{F}_q$ for

¹⁴The description of non-binary BP decoding in this section follows the one in [BMSV15].

efficiently computing the check node operations (see also [BD03]). Its drawback is that the implementation needs to be carried out in the probability domain and thus is not well suited for an actual deployment due to potential numerical instabilities. An implementation in the logarithmic domain has been given in [WSM04], which, however, does not make use of the Fourier transform for a low-complexity computation of the check node operation (which considerably reduces the complexity for large field sizes q). Here, the approach of [SC03, BT04] is followed which uses an FFT and a representation of messages in the *signed log domain* to carry out the check node update, thereby resulting in exact BP decoding. Other approaches only approximate BP decoding and instead aim at reducing the complexity by computing only the most relevant terms of the update equation [VDV⁺10], thus, losing some performance compared to the signed log domain algorithm according to [BT04] which is briefly described in the following.

All messages in the message-passing process are represented by non-binary LLR vectors, and all summations and multiplications of LLR vectors are performed element-wise. As described above, the binary channel LLRs are initially mapped to the non-binary LLR vectors $\mathbf{L}_n$, $n \in \{1, \dots, N_q\}$ by (2.91).

The message vector sent from variable node n to check node m is denoted $\mathbf{Q}_{n,m}$ and the *a posteriori* LLR vector of the n -th symbol is denoted $\mathbf{Q}_n$. Both the variable-to-check messages and the *a posteriori* messages are initialized with the channel output $\mathbf{L}_n$, for $n \in \{1, \dots, N_q\}$. The messages $\mathbf{R}_{m,n}$ which are passed from check node m to variable node n are initialized with zeros.

While the check node update is given by a single equation (2.41) for the binary case, the non-binary check node update consists of multiple steps. First, the elements of each variable-to-check message vector are permuted according to the check matrix entry $a = H_{m,n} \in \mathbb{F}_q^+$

$$\tilde{\mathbf{Q}}_{n,m} = \mathbf{Q}_{n,m} \cdot \mathbf{P}_a, \quad \forall n \in \{1, \dots, N_q\}, m \in \mathcal{M}(n). \quad (2.92)$$

The $q-1$ permutation matrices $\mathbf{P}_a$, with $a \in \mathbb{F}_q^+$, move the i -th element of $\mathbf{Q}_{n,m}$ to position j in $\tilde{\mathbf{Q}}_{n,m}$, with $j = \log_\alpha(a \cdot \alpha^{i-2}) + 2$ and for $2 \leq i \leq q$. The first element ($i = 1$) always remains in its position. The function $\log_\alpha(\cdot)$ here simply yields the exponent of an $\mathbb{F}_q$ element in exponential representation. The permuted messages are then transformed to the signed log domain [BT04] according to

$$\begin{aligned} \tilde{\varphi}_{n,m} &= (\tilde{\varphi}_{n,m}^s, \tilde{\varphi}_{n,m}^m), \quad \forall n \in \{1, \dots, N_q\}, m \in \mathcal{M}(n) \\ \text{with } \tilde{\varphi}_{n,m}^s &= \underbrace{(1, \dots, 1)}_{q \text{ times}}, \quad \tilde{\varphi}_{n,m}^m = \tilde{\mathbf{Q}}_{n,m}, \end{aligned} \quad (2.93)$$

where the superscript $(\cdot)^s$ denotes the sign and the superscript $(\cdot)^m$ denotes the magnitude of a signed log domain value. To transform these values to the Fourier domain, the q -point FFT $\mathcal{F}(\cdot)$ is applied [BT04]:

$$\tilde{\Phi}_{n,m} = \mathcal{F}(\tilde{\varphi}_{n,m}), \quad \forall n \in \{1, \dots, N_q\}, m \in \mathcal{M}(n). \quad (2.94)$$

Finally, the check node update equations can be written as

$$\tilde{\Theta}_{m,n}^s = \prod_{n' \in \mathcal{N}(m) \setminus \{n\}} \tilde{\Phi}_{n',m}^s \quad (2.95a)$$

$$\text{and} \quad \tilde{\Theta}_{m,n}^m = \sum_{n' \in \mathcal{N}(m) \setminus \{n\}} \tilde{\Phi}_{n',m}^m, \quad \forall m \in \{1, \dots, M_q\}, n \in \mathcal{N}(m). \quad (2.95b)$$

Before the resulting message is sent back to a variable node, the inverse Fourier transform

$$\tilde{\theta}_{m,n} = \mathcal{F}^{-1}(\tilde{\Theta}_{m,n}), \quad \forall m \in \{1, \dots, M_q\}, n \in \mathcal{N}(m) \quad (2.96)$$

is computed, the magnitude is extracted from the signed log domain

$$\tilde{\mathbf{R}}_{m,n} = \tilde{\theta}_{m,n}^m, \quad \forall m \in \{1, \dots, M_q\}, n \in \mathcal{N}(m), \quad (2.97)$$

and the inverse permutation (with $a = H_{m,n} \in \mathbb{F}_q^+$)

$$\mathbf{R}_{m,n} = \tilde{\mathbf{R}}_{m,n} \cdot \mathbf{P}_a^{-1}, \quad \forall m \in \{1, \dots, M_q\}, n \in \mathcal{N}(m) \quad (2.98)$$

is applied to the message vector, which concludes the non-binary check node update.

The variable node update is executed according to

$$\mathbf{Q}_{n,m} = \mathbf{L}_n + \left(\sum_{m' \in \mathcal{M}(n) \setminus \{m\}} \mathbf{Q}_{m',n} \right) - \beta_{n,m}, \quad \forall n \in \{1, \dots, N_q\}, m \in \mathcal{M}(n). \quad (2.99)$$

The normalization constant $\beta_{n,m} = \max_{a \in \mathbb{F}_q} Q_{n,m}^{[a]}$ helps to avoid numerical problems. Furthermore, the *a posteriori* LLR vector

$$\mathbf{Q}_n = \mathbf{L}_n + \sum_{m \in \mathcal{M}(n)} \mathbf{R}_{m,n}, \quad \forall n \in \{1, \dots, N_q\} \quad (2.100)$$

is computed for each variable node and a hard decision is performed according to

$$\hat{y}_n = \arg \max_{a \in \mathbb{F}_q} Q_n^{[a]}, \quad \forall n \in \{1, \dots, N_q\}. \quad (2.101)$$

Based on this hard decision, the parity check equations $\hat{\mathbf{y}}\mathbf{H}^\top$ are evaluated at the end of each iteration. The check and variable node updates are executed until all check equations are satisfied or the maximum iteration counter $I_{\max}$ is reached. At the end of the decoding process, the non-binary values in $\hat{\mathbf{y}}$ are expanded to their binary representation as described in Section 2.10.1, yielding the binary decoder output vector $\hat{\mathbf{y}}^{\text{bin}}$.

2.10.3 Ensemble Optimization and Code Construction

In principle, the same methods as introduced for binary LDPC codes can also be used for the optimization of non-binary LDPC codes with only some required modifications. For the ensemble optimization, the DE algorithm needs to be modified to correctly model the behavior of the non-binary BP decoder [RU05, LFK09, WZZ⁺11]. Using an implementation of this non-binary DE, the problem formulation for finding an optimized degree distribution pair follows the one given in Section 2.7.2 and can be solved using methods similar to those developed in Section 2.7.2. It was shown that for increasing field order q , the average variable and check node degrees of optimized degree distributions tend towards smaller values [DM98a, DM98b]. In fact, for field orders of about $q \geq 64$, regular non-binary LDPC codes with variable node degree 2 are found to show the best error correction performance [DM98a], such that there is no need to further optimize degree distributions for these cases.

For the construction of a non-binary check matrix, besides the locations of non-zero elements also the non-zero values have to be chosen for each position. A noticeable performance gain can already be achieved compared to binary LDPC codes by first constructing a binary check matrix with an algorithm of choice and then replacing each 1 entry by a randomly chosen non-zero element from $\mathbb{F}_q$. More recently, also more sophisticated methods have been proposed for the choice of non-zero values, e.g., [KGP06, PFD08, ZLT⁺08].

As the specific optimization of non-binary LDPC codes is not the focus of this thesis, no further investigation of non-binary code design is conducted here.

Multi-Rate LDPC Codes

Virtually all modern communication systems – most notably in the field of wireless communications – have to deal with (quickly) varying transmission channel conditions. At the same time, the demanded data rate and required strength of error protection strongly depend on the supported services, like, e.g., speech, video, or data transmission. Furthermore, in systems where transmission resources, e.g., radio spectrum, are shared by multiple users, the actual number of active users causes mutual interference and requires an adaptation of data rates and error protection mechanisms.

The FEC functionality of such systems allows for the adaptation to these varying conditions by supporting multiple channel code rates in order to make efficient use of the available resources. The *Long Term Evolution* (LTE) system [3rd10, 3rd08], e.g., uses a rate-1/3 turbo code [BGT93, BG96] and applies bit puncturing to achieve code rates higher than 1/3 and bit repetition to achieve code rates lower than 1/3. In LDPC code based standards like WiMAX, WLAN, or some DVB versions, separate codes are specified for each supported code rate (up to 22 codes in DVB-S2). An exemplary overview is given in Table 3.1. As a consequence, transmitter and receiver have to support all specified codes, which hinders an efficient implementation and leads to increased hardware complexity.

Rather than using separate codes, there exist some alternative techniques for the construction of LDPC codes which are capable of realizing multiple code rates using only a single so-called *mother code*. In [LN02, HKM04, HKKM06] *parity puncturing*

	number of LDPC codes	supported code rates
IEEE 802.16e (WiMAX) [IEE09a]	6	$\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6}$
IEEE 802.11n (WLAN) [IEE09b]	12	$\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6}$
DVB-T2 [Eur09b]	13	$\frac{1}{4}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}$
DVB-S2 [Eur09a]	22	$\frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{8}{9}, \frac{9}{10}$

Table 3.1: Numbers and code rates of LDPC codes specified in different communication standards.

was studied to construct LDPC codes of higher code rates from a low rate mother code. In [LN02, YB04] special approaches of *code extension* were proposed to achieve lower code rates starting from a high rate mother code. [TJ05] was the first publication to propose a combined approach using *information shortening* (which was also termed dummy bit insertion in [XR00]) and parity puncturing to cover code rates 0.1 to 0.9 with a single mother code.

In this thesis, a multi-rate LDPC code is referred to as a code that requires only one encoder and decoder implementation (usually for the mother code) and is capable of realizing multiple code rates. In the following, multi-rate LDPC codes based on a combination of information shortening and parity puncturing as introduced in [TJ05] are investigated. The goal of the investigation is to design a single mother LDPC code that can be used to encode and decode code words of arbitrary target code rates, thus, providing a highly flexible FEC mechanism that can be adapted to different error protection levels on-the-fly. As additional constraint, the rate adaptation must not require any changes to the encoder and decoder hardware, such that the hardware implementation can be fully optimized based only on the mother code and still be used at any of the supported code rates. To achieve this, *information shortening* (Section 3.1) and *parity puncturing* (Section 3.2) are reviewed and novel optimization techniques are presented. After that, a method to jointly employ shortening and puncturing to cover very wide code rate ranges with close-to-optimal error correction performance is developed (Section 3.3). Finally, the performance of the novel scheme is compared to that of several multi-rate schemes from the literature (Section 3.4.1).

3.1 Information Shortening

The technique of *information shortening* inserts known bits (*dummy bits*) into the information bit sequence before encoding with the mother code. This way arbitrary code rates lower than the mother code rate can be realized. In the literature, different terms have been used to describe this technique, including “pruning” (e.g., [CH93]), “code or information shortening” (e.g., [TJ05]), and “dummy bit insertion” (e.g., [XR00]).

After introducing the general approach of information shortening by describing the encoder and decoder for a shortened code, at first, an asymptotic analysis of shortened codes is conducted. The asymptotic effect of information shortening on the ensemble performance of LDPC codes was first analyzed in [TJ05] and later extended to the context of a *Hybrid Automatic Repeat reQuest* (HARQ) scheme in [Oka08]. Additionally, finite length effects of shortening have been dealt with in [LWZ09, BBV11, XLG⁺11, BV13]. The novel techniques pre-published in [BBV11, BV13] will be reviewed in Sections 3.1.3 and 3.1.4.

3.1.1 Encoding and Decoding¹

In Figure 3.1 block diagrams of encoder and decoder using information shortening are shown. The most important property is that the core implementations of the original

¹The description of information shortening in this section follows the one given in [BV13].

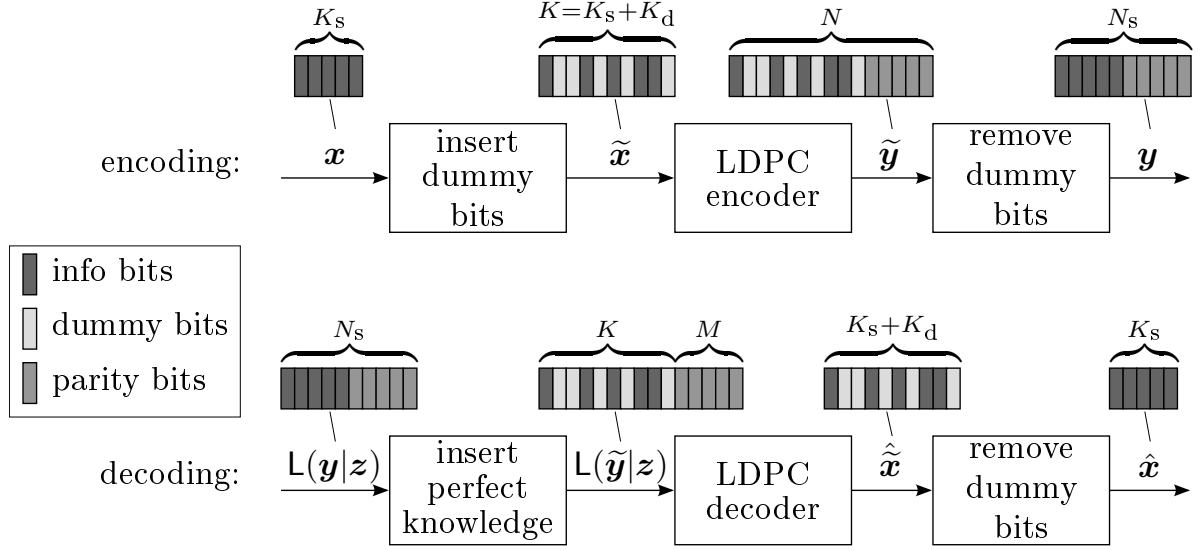


Figure 3.1: Encoding and decoding using a fixed mother encoder and decoder and information shortening to adapt the code rate. The different colors visualize exemplary positions of information, dummy, and parity bits.

LDPC encoder and decoder are left untouched and only some low-complexity pre- and post-processing in the form of inserting or removing constant values is added.

Encoding

At generally arbitrary positions, K_d known dummy bits are inserted into the original information word $\mathbf{x}$ of (shortened) length K_s , yielding the intermediate information word $\tilde{\mathbf{x}}$ of length $K = K_s + K_d$. In the following, all dummy bits are assumed to be set to zero w.l.o.g.. The systematic encoder of the rate $r = \frac{K}{N}$ mother code appends $M = \frac{1-r}{r}(K_s + K_d)$ parity bits. After this, the dummy bits that are included in the intermediate code word $\tilde{\mathbf{y}}$ are removed (*punctured*), since they are known at the receiver and therefore do not need to be transmitted. The resulting code word $\mathbf{y}$ has length $N_s = K_s + M$ and is passed to the modulator for transmission.

While the inner LDPC encoder has input length K , output length N , and code rate r , the resulting system including information shortening has *effective* input length K_s , *effective* output length N_s , and *effective* code rate r_s , which are related by

$$r_s = \frac{K_s}{N_s} = \frac{K_s}{K_s + M} = \frac{K_s}{K_s + \frac{1-r}{r}(K_s + K_d)} = \frac{k_s}{k_s + \frac{1-r}{r}}, \quad (3.1)$$

where the relative shortened information word and code word lengths are defined as k_s and n_s by

$$k_s := \frac{K_s}{K} = \frac{r^{-1} - 1}{r_s^{-1} - 1}, \quad (3.2)$$

$$n_s := \frac{N_s}{N} = \frac{1-r}{1-r_s}. \quad (3.3)$$

Any code rate $0 < r_s \leq r$ can be achieved by an accordant choice of k_s (of course only in discrete steps, depending on the absolute values of K_s and K). If no dummy bits are used (i.e., $k_s = 1$) $r_s = r$ follows, and if the number of information bits approaches zero (i.e., $k_s \rightarrow 0$), the effective code rate also approaches zero.

Decoding

At the receiver, the dummy bits and their positions are known and thus provide perfect *a priori* information which is inserted into the information positions of the received sequence $\mathbf{L}(\mathbf{y}|\mathbf{z})$ of LLRs in form of values $+\infty$ (according to the known bit value of 0) yielding the LLR sequence $\mathbf{L}(\tilde{\mathbf{y}}|\mathbf{z})$. Then, straightforward LDPC decoding of this padded sequence of LLRs is conducted. Finally, the dummy bits are removed from the output.

The positions with perfect knowledge influence the message-passing decoding as follows. If a parity check equation of the form $y_1 + y_2 + \dots + y_\eta = 0$ is considered, in which one or more dummy bits participate that are known to be 0, the equation can be equivalently described by simply discarding those bits from the equation, due to $y_i + 0 = y_i$. An analogous equivalence also holds for the computation of messages in the LLR domain during a check node update according to (2.41), since $L_i \boxplus +\infty = L_i$. Consequently, the outgoing messages of variable nodes that are associated with dummy bits (these nodes are also denoted *dummy nodes* in the following) do not have to be considered in any check node update since their values are $+\infty$. Since there is no need for improving the reliability of a dummy bit's (known) value, also the incoming messages of dummy nodes are of no particular use. According to this observation, all edges connected to dummy nodes can be removed from the Tanner graph without any influence on the decoding process. Hence, the effective graph as "seen" by the decoder is the graph of the mother code with all dummy nodes (and all connected edges) simply removed. Accordingly, the effective parity check matrix is achieved by removing the columns associated with dummy bit positions from the mother code's parity check matrix.

As a consequence, even though the code length of the employed mother code is N , the complexity of decoding is equivalent to that of decoding an LDPC code with smaller code length N_s and code rate r_s if the dummy nodes are simply deactivated. However, it is not mandatory to remove any nodes or edges at all but it just serves reducing complexity. Existing decoder implementations do not have to be changed in any way and are fully compatible with this rate adaptation technique.

It should further be noted that due to the correspondence of each code word position to a check matrix column, encoding and decoding with arbitrary positions of dummy bits can be simplified. Instead of inserting dummy bits at the desired predefined positions before each encoding and decoding process, the check matrix columns can simply be reordered once in advance such that dummy bits can always be appended, e.g., at the beginning or the end of the information word.

3.1.2 Asymptotic Analysis and Optimization

As described above, decoding of a shortened LDPC code is equivalent to decoding in the Tanner graph with all dummy nodes (and associated edges) removed. For the asymptotic analysis of a shortened code of rate r_s , the shortened degree distribution pair (in node perspective) of this pruned graph is of interest and is denoted

$$\Lambda^{[rs]}(x) = \sum_{i=2}^{d_v} \Lambda_i^{[rs]} x^i \quad \text{and} \quad R^{[rs]}(x) = \sum_{j=2}^{d_c} R_j^{[rs]} x^j, \quad (3.4)$$

in analogy to the un-shortened degree distributions $\Lambda(x)$ and $R(x)$ from (2.21).

Shortened Variable Node Degree Distribution

The number of variable nodes of degree i before shortening is given by

$$N_i \stackrel{(2.83)}{=} \Lambda_i N,$$

while the number of variable nodes of degree i after shortening is

$$N_{s,i} = \Lambda_i^{[rs]} N_s = \Lambda_i^{[rs]} n_s N \quad (3.5)$$

The original variable node degree distribution $\Lambda(x)$ and the shortened variable node degree distribution $\Lambda^{[rs]}(x)$ can, thus, be related by the degree distribution $\Delta^{[rs]}(x)$ of all dummy nodes by

$$\Delta^{[rs]}(x) = \sum_{i=2}^{d_v} \Delta_i^{[rs]} x^i = \frac{1}{N - N_s} \sum_{i=2}^{d_v} (N_i - N_{s,i}) x^i \quad (3.6)$$

where $N_i - N_{s,i}$ corresponds to the number of degree- i dummy nodes and $N - N_s$ corresponds to the total number of dummy nodes. This equation can be further rewritten as

$$\begin{aligned} \Delta^{[rs]}(x) &= \sum_{i=2}^{d_v} \left(\frac{\Lambda_i N - \Lambda_i^{[rs]} n_s N}{(1 - n_s) N} \right) x^i \\ &= \sum_{i=2}^{d_v} \left(\Lambda_i \frac{1 - r_s}{r - r_s} - \Lambda_i^{[rs]} \frac{1 - r}{r - r_s} \right) x^i \end{aligned} \quad (3.7)$$

$$\begin{aligned} \Leftrightarrow \quad \Lambda_i^{[rs]} &= \frac{\Lambda_i N - \Delta^{[rs]}(N - N_s)}{N_s}, \\ &= \Lambda_i \frac{1 - r_s}{1 - r} - \Delta_i^{[rs]} \frac{r - r_s}{1 - r}, \quad i \in \{2, \dots, d_v\}. \end{aligned} \quad (3.8)$$

Given a fixed mother LDPC code, the shortened variable node degree distribution can be controlled by (3.8) and by choosing the dummy node degree distribution $\Delta^{[rs]}(x)$, which is realized by inserting the dummy bits at positions with the respective target

degrees. Of course, there are constraints regarding the values of the $\Lambda_i^{[rs]}$, as, e.g., no new variable node degrees can be created by only removing variable nodes from a graph. These constraints can be expressed as bounding the coefficients $\Delta^{[rs]}$ of the dummy node degree distribution between zero and one:

$$\begin{aligned}
0 &\leq \Delta_i^{[rs]} \stackrel{(3.8)}{=} \Lambda_i \frac{1-r_s}{r-r_s} - \Lambda_i^{[rs]} \frac{1-r}{r-r_s} \\
\Leftrightarrow N_{s,i} &\leq N_i \\
\Leftrightarrow \Lambda_i^{[rs]} &\leq \Lambda_i \frac{1-r_s}{1-r}, \quad i \in \{2, \dots, d_v\}
\end{aligned} \tag{3.9}$$

and

$$\begin{aligned}
1 &\geq \Delta_i^{[rs]} \stackrel{(3.8)}{=} \Lambda_i \frac{1-r_s}{r-r_s} - \Lambda_i^{[rs]} \frac{1-r}{r-r_s} \\
\Leftrightarrow N - N_s &\geq N_i - N_{s,i} \\
\Leftrightarrow \Lambda_i^{[rs]} &\geq \Lambda_i \frac{1-r_s}{1-r} - \frac{r-r_s}{1-r}, \quad i \in \{2, \dots, d_v\}.
\end{aligned} \tag{3.10}$$

Together with the trivial constraint of $0 \leq \Lambda_i^{[rs]} \leq 1$ this yields

$$\max \left(0, \Lambda_i \frac{1-r_s}{1-r} - \frac{r-r_s}{1-r} \right) \leq \Lambda_i^{[rs]} \leq \min \left(1, \Lambda_i \frac{1-r_s}{1-r} \right), \quad i \in \{2, \dots, d_v\}. \tag{3.11}$$

Shortened Check Node Degree Distribution

The shortened check node degree distribution $R^{[rs]}$ strongly depends on how dummy nodes are selected. If no special attention is paid to the check node degrees during selection, a certain randomness can be expected in the resulting check node degree distribution. To quantify this randomness, at first, the probability of an arbitrary edge in the original graph to be removed after shortening is considered:

$$\begin{aligned}
P_{\text{removed}} &= P(\text{"edge is removed"}) = P(\text{"edge connected to dummy node"}) \\
&= \sum_{i=2}^{d_v} \underbrace{P(\text{"edge connected to degree } i \text{ variable node"})}_{=\lambda_i} \underbrace{P(\text{"degree } i \text{ variable node is dummy node"})}_{=\frac{\# \text{ degree } i \text{ dummy nodes}}{\# \text{ degree } i \text{ variable nodes}}} \\
&= \sum_{i=2}^{d_v} \lambda_i \frac{\Delta_i^{[rs]}(N - N_s)}{\Lambda_i N} = \sum_{i=2}^{d_v} \lambda_i \frac{\Lambda_i N - \Lambda_i^{[rs]} N_s}{\Lambda_i N} \\
&= \sum_{i=2}^{d_v} \lambda_i \left(1 - \frac{\Lambda_i^{[rs]} N_s}{\Lambda_i N} \right)
\end{aligned}$$

$$\begin{aligned}
&= 1 - \frac{N_s}{N} \sum_{i=2}^{d_v} \frac{\lambda_i}{\Lambda_i} \Lambda_i^{[rs]} \\
&\stackrel{\text{with (2.22)}}{=} 1 - \frac{N_s}{N} \frac{\sum_{i=2}^{d_v} i \Lambda_i^{[rs]}}{\sum_{i=2}^{d_v} i \Lambda_i} = 1 - \frac{E^{[rs]}}{E},
\end{aligned} \tag{3.12}$$

where E denotes the number of edges in the original graph and $E^{[rs]}$ the number of edges in the shortened graph. The number of remaining edges of a degree- k check node (cn) after shortening then follows a binomial distribution with

$$\begin{aligned}
&\text{P}(\text{“cn has } j \text{ edges after shortening”} \mid \text{“cn has } k \text{ edges before shortening”}) \\
&= \binom{k}{j} (1 - \text{P}_{\text{removed}})^j \text{P}_{\text{removed}}^{k-j},
\end{aligned} \tag{3.13}$$

yielding the probability of a degree- j check node after shortening by marginalizing:

$$\begin{aligned}
R_j^{[rs]} &= \text{P}(\text{“cn has } j \text{ edges after shortening”}) \\
&= \sum_{k=2}^{d_c} \text{P}\left(\text{“cn has } j \text{ edges after shortening”} \mid \text{“cn has } k \text{ edges before shortening”}\right) \underbrace{\text{P}\left(\text{“cn has } k \text{ edges before shortening”}\right)}_{=R_k} \\
&= \sum_{k=2}^{d_c} R_k \binom{k}{j} (1 - \text{P}_{\text{removed}})^j \text{P}_{\text{removed}}^{k-j}.
\end{aligned} \tag{3.14}$$

The shortened check node degree distribution is finally given by

$$\begin{aligned}
R^{[rs]}(x) &= \sum_{j=0}^{d_c} R_j^{[rs]} x^j \\
&= \sum_{j=0}^{d_c} \left(\sum_{k=2}^{d_c} R_k \binom{k}{j} (1 - \text{P}_{\text{removed}})^j \text{P}_{\text{removed}}^{k-j} \right) x^j.
\end{aligned} \tag{3.15}$$

This degree distribution potentially contains degree-0 and degree-1 check nodes (depending on the original check node degree distribution and on how many nodes are shortened) which are undesirable, as they do not serve any purpose during message-passing decoding. To avoid these degree 0 and 1 check nodes, an easy strategy is to simply not choose a variable node as dummy node if this causes one or more check nodes to reduce their degree to below 2.

Threshold Computation

The asymptotic error correction performance of a shortened code of effective rate r_s is determined by the shortened degree distribution pair $(\Lambda^{[rs]}, R^{[rs]})$. For the asymptotic case (i.e., at infinite code length), it does not make a difference whether a code from the $(\Lambda^{[rs]}, R^{[rs]})$ ensemble was constructed directly or by shortening a code from another ensemble. Thus, the performance of shortened LDPC codes can be evaluated using the methods introduced for LDPC code ensembles in Section 2.6.1 by simply considering the shortened degree distribution pairs.

Constrained Optimization

Given a fixed mother code degree distribution pair (λ, ρ) with corresponding node perspective (Λ, R) and code rate r , the asymptotic performance of a shortened code of code rate r_s can be optimized by minimizing the ensemble threshold $\tau(\lambda^{[rs]}, \rho^{[rs]})$ of the shortened degree distribution pair $(\lambda^{[rs]}, \rho^{[rs]})$ with node perspective $(\Lambda^{[rs]}, R^{[rs]})$. The optimization problem is given by

$$\min \quad \tau(\lambda^{[rs]}, \rho^{[rs]}) \quad (3.16a)$$

$$\text{subject to} \quad \lambda^{[rs]}(1) = \sum_{i=2}^{d_v} \lambda_i^{[rs]} = 1 \quad (3.16b)$$

$$\lambda_i^{[rs]} \geq 0, \quad i = 2, \dots, d_v \quad (3.16c)$$

$$\Lambda_i \frac{1-r_s}{1-r} - \frac{r-r_s}{1-r} \stackrel{(3.11)}{\leq} \Lambda_i^{[rs]} \stackrel{(3.11)}{\leq} \Lambda_i \frac{1-r_s}{1-r}, \quad i = 2, \dots, d_v \quad (3.16d)$$

$$R^{[rs]}(x) \text{ from (3.15)} \quad (3.16e)$$

$$\lambda_2^{[rs]} \stackrel{(2.50)}{<} \frac{e^{r\tau(\lambda^{[rs]}, \rho^{[rs]})}}{\sum_{j=2}^{d_c} \rho_j^{[rs]}(j-1)}, \quad (3.16f)$$

where the trivial constraints (3.16b) and (3.16c) ensure $\lambda^{[rs]}$ to be a valid degree distribution and (3.16d) resembles the shortening specific constraint introduced by (3.11). Additionally, the stability constraint (2.50) is included as constraint (3.16f). Finally, a random selection of the shortened columns of the mother code matrix is assumed in constraint (3.16e), resulting in the check node degree distribution given by (3.15). If the check node degree distribution $R(x)$ of the mother code is concentrated (as defined by (2.73)), the column selection strategy can be tailored in such a way that the shortened check node degree distribution $R^{[rs]}(x)$ is also (nearly) concentrated (see Section 3.3.2). This column selection strategy can be incorporated into the optimization problem by replacing constraint (3.16e) by the check concentration constraint

$$\rho^{[rs]} \text{ and } \bar{d}_c^{[rs]} \text{ from (2.73) and (2.74).} \quad (3.17)$$

For both cases (randomly shortened check node degree distribution or concentrated shortened check node degree distribution), the global optimization using DDE or the linearization using GADE as described in Section 2.7.2 can in principle be used. It should, however, be considered that the GADE approach does not give very reliable results for code rates lower than about 0.5.

This constrained optimization technique was already used in [TJ05] to successively optimize shortened variable node degree distributions for multiple code rates, starting from a fixed mother code of rate 0.5. These reference results will be presented later in Section 3.4.1.

3.1.3 Finite Length Optimization: Maximum Distance²

Besides the asymptotic performance described above, also finite length effects of information shortening are of interest. In this context, finite length effects are those that are not related to the degree distributions but to the concrete structure of a check matrix of finite size. One such finite length optimization based on a maximum distance optimization was introduced in [BBV11] and is reviewed in the following.

Instead of simply appending dummy bits to the information bits or selecting the dummy bit positions randomly, this approach determines optimized dummy bit positions within the information bit sequence. The approach is inspired by the argumentation of [BEV11], where dummy bits are inserted equidistantly into the information bit vector before convolutional encoding. While convolutional codes introduce strong dependencies between neighboring bits, the relation between code bits of an LDPC code is not determined by the positions within a frame but solely by the connections of nodes in the Tanner graph. Thus, distributing variable nodes with perfect knowledge (i.e., dummy nodes) throughout the Tanner graph as uniformly as possible is not as trivial as for convolutional codes. Since the proposed method selects optimized dummy bit positions of a fixed given LDPC code, it can be directly incorporated into existing systems to add flexibility in terms of the supported code rates.

The following greedy heuristic is proposed to maximize the average minimum pairwise distance (i.e., the length of the shortest path) between a given number of K_d dummy nodes which are to be selected in a given Tanner graph.³ First, a simple breadth first search is applied to find the minimum pairwise distances between all K information nodes. The result is given by the $K \times K$ distance matrix $\mathbf{D} \in \mathbb{N}^{K \times K}$, where matrix entry $D_{i,j}$ denotes the minimum distance between information node i and information node j . The distance matrix $\mathbf{D}$ is symmetric and has zeros on the main diagonal, since the Tanner graph is undirected. After the distance matrix has been determined, K_d information nodes are selected and added to the set of dummy nodes $\mathcal{V}_d$ in a node-by-node manner. The first two nodes that are added to $\mathcal{V}_d$ are chosen as $i_{\max}$ and $j_{\max}$ with

$$(i_{\max}, j_{\max}) \in \arg \max_{\forall i,j} D_{i,j}, \quad (3.18)$$

i.e., two information nodes that are most distant from each other. From the third node on, the node to be selected next is the information node i^* that maximizes the average distance to all information nodes j that have already been selected:

$$i^* \in \arg \max_{\forall i \notin \mathcal{V}_d} \frac{1}{|\mathcal{V}_d|} \sum_{\forall j \in \mathcal{V}_d} D_{i,j} = \arg \max_{\forall i \notin \mathcal{V}_d} \sum_{\forall j \in \mathcal{V}_d} D_{i,j}, \quad (3.19)$$

where $|\mathcal{V}_d|$ denotes the cardinality of $\mathcal{V}_d$, and thus the number of dummy nodes already selected. The algorithm stops when K_d nodes have been added to $\mathcal{V}_d$ (i.e., $|\mathcal{V}_d| = K_d$). A summary is given in Algorithm 2.

²The description in this section follows the pre-published paper [BBV11].

³To avoid dummy nodes that are very close in the Tanner graph, rather than maximizing the average distance, an alternative approach that maximizes the minimum distance can be employed, but is not further followed up upon here.

Algorithm 2 Maximum Distance Dummy Bit Insertion (MD-DBI).

-
- 1: Compute distance matrix $\mathbf{D}$ with distances between all K information nodes
 - 2: Choose randomly two most distant nodes $(i_{\max}, j_{\max}) \in \arg \max_{\forall i,j} D_{i,j}$
 - 3: Initialize $\mathcal{V}_d = \{i_{\max}, j_{\max}\}$
 - 4: **while** $|\mathcal{V}_d| < K_d$ **do**
 - 5: Choose randomly $i^* \in \arg \max_{\forall i \notin \mathcal{V}_d} \sum_{j \in \mathcal{V}_d} D_{i,j}$
 - 6: $\mathcal{V}_d = \mathcal{V}_d \cup \{i^*\}$
 - 7: **end while**
-

When rate-adaptation is used, the code rate r_s (and thus the number K_d of dummy bits) is typically not fixed but possibly varies from one frame to the next. However, in this case it is still sufficient to run the proposed algorithm once using the maximum possible number of dummy bits $K_d = K$ and keep track of the order in which dummy nodes are added to $\mathcal{V}_d$, resulting in an ordered list of potential dummy bit positions of length K . For each requested code rate r_s smaller than the original code rate r , the corresponding number of dummy bit positions can simply be obtained as the first K_d entries of this list.

In Figure 3.2 an exemplary Tanner graph is shown with five parity nodes and five information nodes, three of which are used for dummy bits. In the left example, the dummy bit positions are simply chosen as the last three information bit positions resulting in an average pairwise distance of 2.67 edges between dummy nodes. In the right example, on the other hand, the first, second, and fourth node are selected as dummy nodes according to the proposed algorithm, resulting in an increased average pairwise distance of 4 edges.

Simulation Results

To evaluate the performance of the proposed optimized dummy bit insertion, the LDPC codes specified in the IEEE 802.16e WiMAX standard [IEE09a] are considered. Belief Propagation decoding as described in Section 2.5 with a maximum of $I_{\max} = 200$ iterations is conducted. The following three setups are considered:

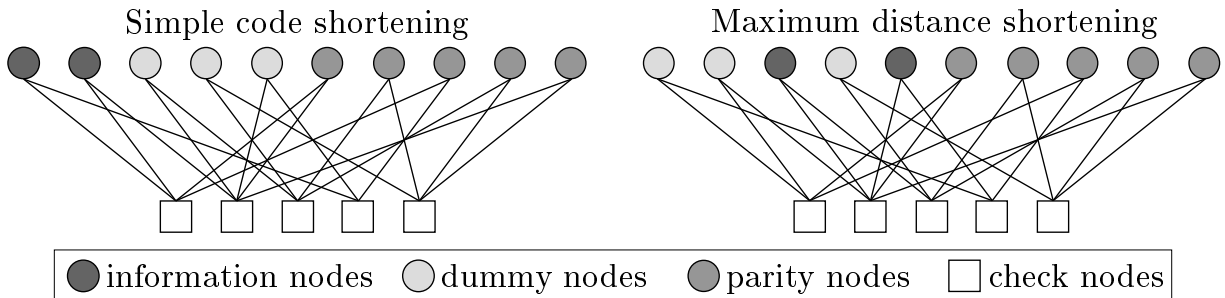


Figure 3.2: Exemplary Tanner graph with three shortened nodes. Simple code shortening (left) results in the average pairwise distance 2.67. The proposed maximum distance shortening (right) results in the increased average pairwise distance of 4.

Reference codes (Ref.): The codes from the WiMAX standard with code length $N = 576$ and code rates $r \in \{\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6}\}$.

Random dummy bit insertion (R-DBI): Dummy bits are randomly chosen amongst the information positions. Rate $r = \frac{5}{6}$ WiMAX codes of code length $N_s = 576$ after shortening are used as mother codes and 40 %, 60 %, and 80 % of the available information bits are dummy bits in order to match the shortened code rates $r_s = \frac{3}{4}$, $r_s = \frac{2}{3}$, and $r_s = \frac{1}{2}$, respectively.

Maximum distance dummy bit insertion (MD-DBI): Same as for the setup R-DBI, but dummy bits are now inserted at optimized positions according to the maximum distance heuristic from Algorithm 2.

The code lengths N of the mother codes for the R-DBI and MD-DBI setups are chosen to result in the same shortened code length $N_s = n_s N = 576$ of the transmitted frame as for the reference codes (i.e., $N = 864$ for $r_s = \frac{3}{4}$, $N = 1152$ for $r_s = \frac{2}{3}$ and $N = 1728$ for $r_s = \frac{1}{2}$). While this setup does not allow a rate-adaptation to all code rates with the same mother code, it is chosen to provide a fair comparison of the resulting shortened codes to the original WiMAX codes of code length $N = 576$. This way the simulation results for each shown code rate also exhibit the same code length such that the curves are not biased due to different code lengths but only show the influence of the different rate-adaptation methods.

Note that despite the increased code lengths of the codes using dummy bit insertion, decoding complexity is not increased since all additional nodes and edges of the Tanner graph are associated with dummy bits. Thus, as mentioned above in Section 3.1.1, these nodes and edges do not have to be considered during decoding, since they hold messages with values $+\infty$, which is the identity element of the boxplus operator.

The upper plot in Figure 3.3 shows the frame error rate performance over the channel quality E_s/N_0 in dB. For a shortened code rate of $r_s = \frac{3}{4}$, the waterfall behavior of all three setups is nearly identical. Yet, the error floor performance is slightly degraded for the non-optimized setup R-DBI and slightly enhanced for the optimized case MD-DBI. Except for a small loss in the waterfall region, a similar behavior is observed for a shortened code rate of $r_s = \frac{2}{3}$. The most evident difference between setups R-DBI and MD-DBI can be seen for the code rate $r_s = \frac{1}{2}$. For the non-optimized case R-DBI, the error floor already starts at a frame error rate of about 10^{-2} , in contrast to the optimized case MD-DBI which comes very close to the original rate-1/2 code with a loss of less than 0.2 dB near the waterfall and even less in the error floor region.

In the three lower plots of Figure 3.3, the convergence behavior is depicted for the three different setups. For each shortened code rate r_s , the frame error rate is plotted over the number of decoding iterations for exemplary fixed E_s/N_0 values. For $r_s = \frac{1}{2}$ and $E_s/N_0|_{\text{dB}} = 0.99$ dB (left plot), it can be seen that convergence is much slower for setup R-DBI than for the reference code and setup MD-DBI. The setup MD-DBI, however, converges even faster than the original rate-1/2 code, but achieves a slightly higher frame error rate if the number of decoding iterations is high. For $r_s = \frac{2}{3}$ and $E_s/N_0|_{\text{dB}} = 2.64$ dB (center plot), a very similar behavior is observed. For $r_s = \frac{3}{4}$ and $E_s/N_0|_{\text{dB}} = 3.75$ dB (right plot), the original rate-3/4 code exhibits

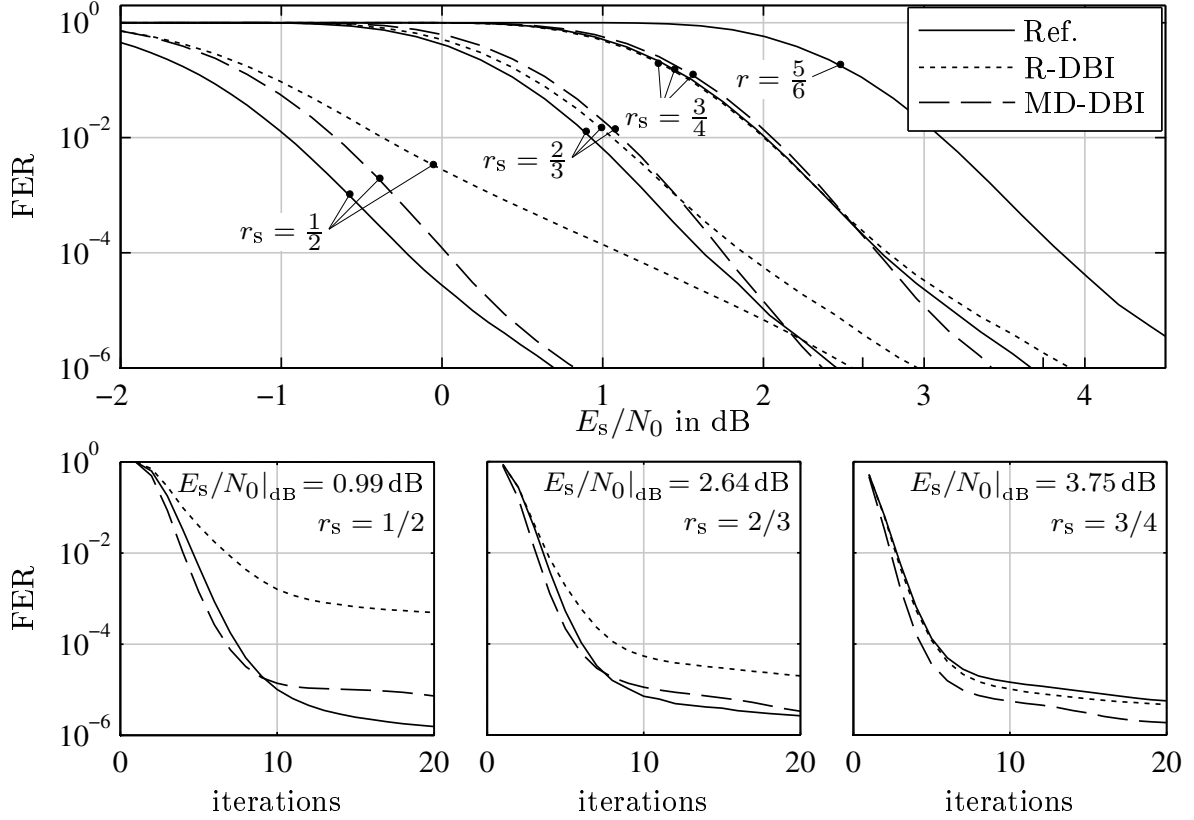


Figure 3.3: FER over channel quality E_s/N_0 after $I_{\max} = 200$ iterations is shown in the upper plot. The convergence (FER over iterations) at fixed E_s/N_0 values for the three shortened code rates is shown in the lower plots.

the slowest convergence of all setups but still converges to a lower frame error rate than for setup R-DBI. The optimized dummy bit insertion (MD-DBI), however, has the fastest convergence and also achieves the best frame error rate for small as well as for high numbers of iterations.

In summary, the proposed optimization ensures that the perfect knowledge provided by known dummy bits is evenly distributed throughout the code's Tanner graph. Simulation results show that significant gains in terms of frame error rate performance and convergence speed can be achieved by this approach as compared to non-optimized insertion of dummy bits at the end of the payload. It should be noted, however, that no care was taken at all to control the shortened degree distribution in the conducted experiments. Of course, this finite length optimization can be combined with the asymptotic optimization presented in Section 3.1.2 by modifying the algorithm to choose a corresponding number of dummy nodes for each degree to also achieve a good degree distribution after shortening.

3.1.4 Finite Length Optimization: Breaking Cycles⁴

Another finite length optimization termed *breaking cycles* is presented in this section. It has been pre-published in [BV13] and is based on the idea of exploiting dummy

⁴The description in this section follows the pre-published paper [BV13].

nodes for “breaking” short cycles to improve the message-passing decoding performance. Similarly to the maximum distance optimization of the previous section, the presented approach is applicable to arbitrary LDPC codes and can thus be easily integrated into existing systems.

As described before, from a message-passing decoder’s point of view, inserting dummy bits means removing edges from the Tanner graph. The breaking cycles method basically chooses the positions for the dummy bits such that the removal of edges “breaks” as many shortest cycles as possible. More precisely, since removing a node also removes all cycles running through it, the nodes chosen for dummy bits are the ones with most cycles running through them.

Naturally, this approach will tend to select many high degree nodes, since these are more likely to have a high number of cycles. If the resulting shortened graph is desired to have a specific predetermined variable node degree distribution, this can easily be achieved by a minor modification of the proposed approach (i.e., pre-compute the required number of dummy bits for each degree and stop choosing these degrees as soon as the target quantity is reached). However, this approach solely focuses on the removal of cycles and the effect on the degree distribution is disregarded for the moment (the influence of a modified degree distribution will also be dealt with in the simulation results on page 64).

In [KB10, KB13], an efficient message-passing algorithm for counting the number of cycles of length up to $2g - 2$ was presented, with g being the girth, i.e., the length of the shortest cycle in the graph. The complexity of the algorithm grows as $\mathcal{O}(gE^2) = \mathcal{O}(N^2)$ (with E denoting the number of edges in the Tanner graph) which is only feasible for moderate code lengths. Here, a slight modification of this algorithm is used that does not count the global number of cycles, i.e., the cycle histogram $\boldsymbol{\alpha} = (\alpha_4, \alpha_6, \dots, \alpha_{2g-2})$, but the local cycle histogram $\boldsymbol{\alpha}_n = (\alpha_{n,4}, \alpha_{n,6}, \dots, \alpha_{n,2g-2})$ of each information node $n \in \mathcal{I}$ (i.e., each potential candidate for a dummy node).

A Tanner graph $\mathcal{G} = (\mathcal{I} \cup \mathcal{P} \cup \mathcal{C}, \mathcal{E})$ with the set of information nodes $\mathcal{I}$, the set of parity nodes $\mathcal{P}$, the set of check nodes $\mathcal{C}$, and the set of connecting edges $\mathcal{E}$ is considered. The proposed algorithm selects information nodes from $\mathcal{I}$ to be used for dummy bits in a node-by-node manner until all K information nodes have been selected to identify the order in which dummy bits should be allocated. In each step, the selected node n^* is the one with the largest number of cycles $\alpha_{n^*,g_{\mathcal{I}}}$ of the currently shortest length $g_{\mathcal{I}}$ in the set $\mathcal{I}$.⁵ After node n^* has been removed from the graph, the numbers in the local cycle histograms $\boldsymbol{\alpha}_n$ of other nodes are potentially reduced, and the shortest cycle length $g_{\mathcal{I}}$ might increase. Thus, $g_{\mathcal{I}}$ and $\alpha_{n,g_{\mathcal{I}}}$ have to be recomputed for all remaining nodes n after each step. If a short cycles exists that is only connected to parity nodes, it is theoretically possible that at some stage the value $g_{\mathcal{I}}$ becomes larger than $2g - 2$ (with g still denoting the current global girth). In this case, the values $\alpha_{n,g_{\mathcal{I}}}$ can no longer be computed by the algorithm from [KB10, KB13]. However, in practice this only happens after almost all K nodes from the set $\mathcal{I}$ have already been

⁵Cycles that are shorter than $g_{\mathcal{I}}$ but are only connected to parity nodes are not considered, as they cannot be eliminated by information shortening.

selected, so that the few remaining nodes can simply be chosen in some arbitrary fashion. Finally, the algorithm yields the ordered vector $\mathbf{s}$ of size K containing the order in which bits should be shortened. For each mother code, this algorithm has to be executed only once offline to find the vector $\mathbf{s}$. To realize any shortened code rate $r_s < r$, the first $K_d = K - K_s$ positions in $\mathbf{s}$ are then simply chosen as dummy bits. A detailed description is given in Algorithm 3.

Of course, the columns of the parity check matrix can simply be reordered according to $\mathbf{s}$. This allows to achieve the performance gain of the proposed method by appending the desired number of dummy bits to the end (or likewise to the beginning) of the information bits.

One drawback of this approach is its high computational complexity, since the algorithm of [KB10, KB13] for counting the cycles has to be executed K times (line 10 of Algorithm 3), which leads to an overall complexity of $\mathcal{O}(KN^2) = \mathcal{O}(N^3)$. Although the method is mainly intended to be used for moderate code lengths N and only has to be executed once in advance for a given code, this complexity becomes quickly infeasible for code lengths of about a few thousand bits. To deal with this complexity problem, a simplified version of Algorithm 3 is additionally presented which has been pre-published in [BV13]. This *simplified breaking cycles* algorithm only computes the local cycle histograms α_n once in advance and selects dummy bit positions solely based on these pre-computed values. Consequently, the complexity is reduced to only $\mathcal{O}(N^2)$ which is given by the algorithm of [KB10, KB13] for counting the cycles. This simplified version is described in Algorithm 4.

Empirical investigations suggest that for this simplified breaking cycles algorithm,

Algorithm 3 Breaking Cycles algorithm finds optimized order $\mathbf{s}$ of dummy bits.

```

1: Given a Tanner graph  $\mathcal{G} = (\mathcal{I} \cup \mathcal{P} \cup \mathcal{C}, \mathcal{E})$ 
2: Initialize  $\mathcal{I}' = \mathcal{I}$ ,  $\mathcal{E}' = \mathcal{E}$ ,  $\mathcal{G}' = \mathcal{G}$ 
3: Initialize vector of dummy nodes  $\mathbf{s} = ( )$  // empty vector
4: while size( $\mathbf{s}$ ) <  $K$  do
5:   Find the length  $g$  of the shortest cycle in  $\mathcal{G}'$ 
6:   Find the length  $g_{\mathcal{I}'}$  of the shortest cycle running through any information node
      $n \in \mathcal{I}'$ 
7:   if ( $g_{\mathcal{I}'} > 2g - 2$ ) then
8:     break // cycles cannot be found with [KB10]
9:   end if
10:  Compute  $\alpha_{n, g_{\mathcal{I}'}}$  for all  $n \in \mathcal{I}'$ 
11:  Find  $n^* = \arg \max_{n \in \mathcal{I}'} \alpha_{n, g_{\mathcal{I}'}}$ 
12:   $\mathbf{s} = (\mathbf{s}, n^*)$  // append  $n^*$  to  $\mathbf{s}$ 
13:   $\mathcal{I}' = \mathcal{I}' \setminus \{n^*\}$  // remove  $n^*$  from  $\mathcal{I}'$ 
14:   $\mathcal{E}' = \mathcal{E}' \setminus \mathcal{E}_{n^*}$  //  $\mathcal{E}_{n^*}$  denotes all edges connected to variable node  $n^*$ 
15:   $\mathcal{G}' = (\mathcal{I}' \cup \mathcal{P} \cup \mathcal{C}, \mathcal{E}')$ 
16: end while
17: Append all remaining nodes from  $\mathcal{I}'$  to  $\mathbf{s}$ 

```

Algorithm 4 Simplified Breaking Cycles algorithm for finding optimized order $\mathbf{s}$ of dummy bit positions.

```

1: Given a Tanner graph  $\mathcal{G} = (\mathcal{I} \cup \mathcal{P} \cup \mathcal{C}, \mathcal{E})$ 
2: Initialize  $\mathcal{I}' = \mathcal{I}$ 
3: Initialize vector of dummy nodes  $\mathbf{s} = ( )$  // empty vector
4: Find the local girth  $g_n$  for all  $n \in \mathcal{I}$ 
5: Compute  $\alpha_{n,g_n}$  for all  $n \in \mathcal{I}$ 
6: Set  $\alpha_{n,2t} = 0$  for all  $n \in \mathcal{I}$  and all  $t$  with  $2 \leq t < g_n/2$ 
7: while  $\text{size}(\mathbf{s}) < K$  do
8:   Find  $l = \min_{\forall n \in \mathcal{I}'} g_n$ 
9:   Find  $n^* = \arg \max_{\forall n \in \mathcal{I}'} \alpha_{n,l}$ 
10:   $\mathbf{s} = (\mathbf{s}, n^*)$  // append  $n^*$  to  $\mathbf{s}$ 
11:   $\mathcal{I}' = \mathcal{I}' \setminus \{n^*\}$  // remove  $n^*$  from  $\mathcal{I}'$ 
12: end while

```

the amount by which the number of short cycles in the graph is reduced is almost the same as for the more complex Algorithm 3. In Table 3.2 the numbers of cycles are given for an optimized rate-1/2 PEG LDPC code with code length $N = 2304$, $K = 1152$, girth $g = 6$, and degree distributions (in node perspective)

$$\Lambda(x) = 0.45x^2 + 0.37x^3 + 0.03x^4 + 0.15x^{11}$$

and $R(x) = 0.47x^7 + 0.53x^8$.

In the upper part of the table, the global numbers of cycles α_l of lengths $l \in \{4, 6, 8, 10\}$ are shown, while the lower part shows the number of only those cycles that are running through information nodes $\mathcal{I}$ denoted by $\alpha_l^{\mathcal{I}}$. These numbers are of special interest since only these cycles can be eliminated by dummy bits. To compute the numbers $\alpha_l^{\mathcal{I}}$, the numbers of cycles are counted at each node in a node-by-node manner leading to cycles potentially being counted multiple times, which is why the values $\alpha_l^{\mathcal{I}}$ can be larger than the corresponding values α_l . In addition to the cycles of the mother code, the according numbers are also given if dummy bit insertion with $K_d = K - K_s = 768$ is used to achieve $r_s \stackrel{(3.1)}{=} \frac{K_s}{N_s} = \frac{1152-768}{2304-768} = \frac{1}{4}$. Four different approaches of finding optimized dummy bit positions are considered: the maximum distance heuristic from Section 3.1.3 presented in [BBV11], the proposed Algorithm 3 and the proposed simplified Algorithm 4, as well as the approach presented in [LWZ09], which tries to maximize the *extrinsic sum* of the dummy nodes, i.e., preferably shortens low-degree variable nodes that are connected to high-degree check nodes.

While the approach of [LWZ09] and the MD-DBI only reduce the number of cycles of length 6 by about 20–25 %, both breaking cycles approaches achieve a reduction by about 90 %. A similar behavior is observed for the cycles of length 8 and 10. It is worth noting that within the information nodes $\mathcal{I}$ (in the lower part of the table), in fact, the breaking cycles approaches eliminate all cycles of length 6 and more than 99 % of the cycles of lengths 8 and 10.

global cycles	α_4	α_6	α_8	α_{10}
Mother code, $K_d = 0$	0	2 285	141 934	3 379 631
Approach of [LWZ09], $K_d = 768$	0	1 768	101 415	2 202 533
Alg. 2 (MD-DBI), $K_d = 768$	0	1 822	104 556	2 294 903
Alg. 3 (BreakCyc), $K_d = 768$	0	269	9 573	119 791
Alg. 4 (SimpBreakCyc), $K_d = 768$	0	269	10 438	128 547
node wise cycles of all $n \in \mathcal{I}$	$\alpha_4^{\mathcal{I}}$	$\alpha_6^{\mathcal{I}}$	$\alpha_8^{\mathcal{I}}$	$\alpha_{10}^{\mathcal{I}}$
Mother code, $K_d = 0$	0	3 506	281 903	8 414 973
Approach of [LWZ09], $K_d = 768$	0	2 436	183 644	4 989 422
Alg. 2 (MD-DBI), $K_d = 768$	0	2 559	190 889	5 249 099
Alg. 3 (BreakCyc), $K_d = 768$	0	0	1 086	21 745
Alg. 4 (SimpBreakCyc), $K_d = 768$	0	0	1 990	31 124

Table 3.2: Cycles of rate 1/2 PEG LDPC mother code with $N = 2304$.

Simulation Results

To numerically evaluate the performance of the considered shortening method, the rate-1/2 PEG LDPC code with code length $N = 2304$ from above is used as mother code, and dummy bit insertion with $K_d \in \{768, 1024\}$ dummy bits is applied to realize shortened code rates $r_s \in \{\frac{1}{4}, \frac{1}{10}\}$. The following five setups are considered:

- Approach of [LWZ09] ([LWZ09])
- Maximum distance approach from Algorithm 2 (MD-DBI)
- Breaking cycles from Algorithm 3 (BreakCyc)
- Simplified breaking cycles from Algorithm 4 (SimpBreakCyc)
- Reference PEG codes (Reference PEG)

The reference PEG codes (including the used mother code) are optimized for their specific code rates and serve for comparison only. They have code rates $r \in \{\frac{1}{2}, \frac{1}{4}, \frac{1}{10}\}$ and code lengths $N \in \{2304, 1536, 1280\}$, which correspond to the shortened code rates r_s and code lengths N_s achieved by the dummy bit insertion.

Frame Error Rate Results

Belief Propagation decoding as described in Section 2.5 with a maximum of $I_{\max} = 100$ iterations is conducted. In Figure 3.4 the frame error rate performance over the channel quality E_s/N_0 in dB is shown. For $r_s = \frac{1}{4}$ (solid lines) it can be seen that for the approach of [LWZ09] and MD-DBI, there is a performance gap of about 1 dB compared to the optimized PEG code. Both of the breaking cycles approaches, however, perform even a little better than the optimized PEG code in the waterfall region, but show a slight error floor for error rates below 10^{-4} . At an error rate of 10^{-4} , a gain of about 0.9 dB is achieved compared to [LWZ09] and MD-DBI. For $r_s = \frac{1}{10}$,

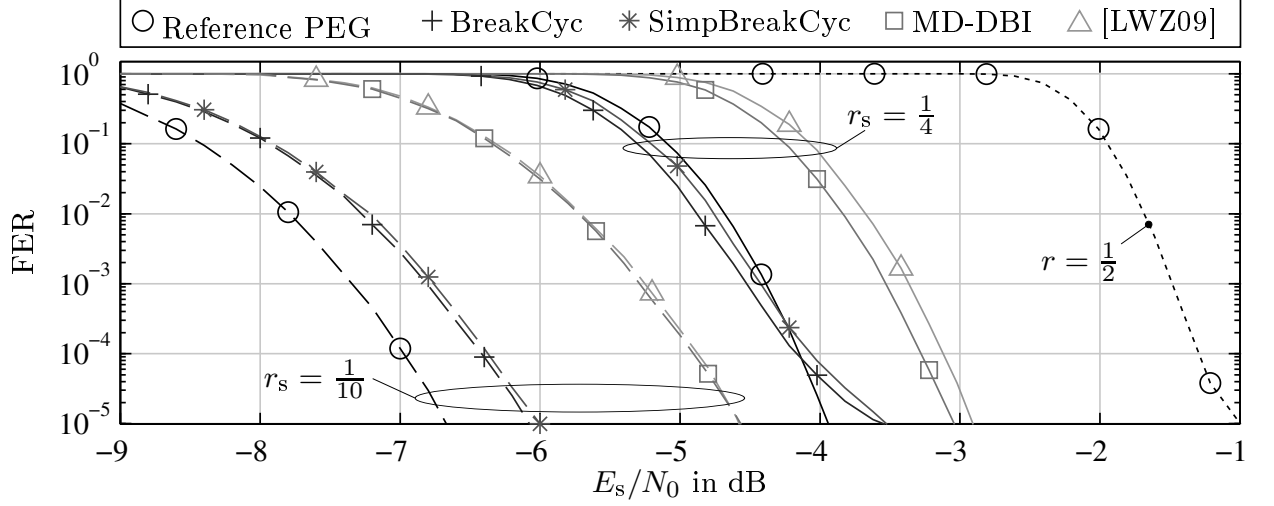


Figure 3.4: FER results for rate- $\frac{1}{2}$ PEG mother code with code length $N = 2304$ and four different approaches of dummy bit insertion. As reference, optimized PEG codes of rates $\frac{1}{4}$ and $\frac{1}{10}$ are also shown.

the breaking cycles approaches show a performance gap of about 0.6 dB compared to the reference PEG code, but still clearly outperform the approach of [LWZ09] and also MD-DBI by about 1.5 dB. Furthermore, the simulation results show that in terms of error correction performance the simplified Algorithm 4 performs very close to Algorithm 3, which complies with the observation that it also reduces the number of short cycles to almost the same extent.

Influence of Degree Distributions

As already indicated above, different algorithms for the selection of dummy bit positions can result in different shortened degree distributions if the allowed degrees of the dummy nodes are not predefined. The variable node degree distributions of the mother code, the shortened rate- $\frac{1}{10}$ code after dummy bit insertion according to [LWZ09] and according to BreakCyc, as well as the optimized rate- $\frac{1}{10}$ PEG code are shown in Table 3.3. The DDE thresholds τ_{DDE} in terms of E_s/N_0 values are computed under the assumption of concentrated check node degree distributions.

Since these varying degree distributions obviously have an influence on the frame error rate results from the previous section, the following additional experiments for

Λ_i or $\Lambda_i^{[rs]}$, $i :$	2	3	4	11	$\tau_{\text{DDE}} _{\text{dB}}$
Mother code, $r = \frac{1}{2}$	0.450	0.370	0.030	0.150	-2.63
[LWZ09], $r_s = \frac{1}{10}$	0.403	0.334	0.028	0.235	-6.42
BreakCyc, $r_s = \frac{1}{10}$	0.492	0.345	0.028	0.135	-8.34
Optimized PEG, $r = \frac{1}{10}$	0.718	0.136	0.098	0.048	-11.03

Table 3.3: Original and shortened variable node degree distributions $\Lambda(x)$ and $\Lambda^{[rs]}(x)$ in node perspective and ensemble thresholds τ_{DDE} with the assumption of concentrated check node degree distributions.

the effective rate r_s are conducted: 20 random selections of dummy bit positions (i.e., random shortening) are instantiated with the same degree distribution as resulting from BreakCyc. Furthermore, by predefining the allowed number of dummy nodes for each degree, the breaking cycles Algorithm 3 is constrained to exactly match the degree distribution as achieved by [LWZ09], denoted by ModBreakCyc. This way, different shortened codes with the same degree distribution can be compared. The results are shown in Figure 3.5.

It can be seen that the BreakCyc achieves better performance than random dummy bit insertion with the same degree distribution. Moreover, also for the degree distribution achieved by the algorithm from [LWZ09] with a much higher threshold, the ModBreakCyc method still outperforms the reference algorithm.

Altogether, the breaking cycles approach is an effective method to find an optimized set of dummy bit positions that shows superior error correction performance for shortening of a given mother code. It can easily be used in conjunction with a predetermined optimized shortened degree distribution to achieve near capacity performance over a wide range of code rates (e.g., 0.5 to 0.1 as presented here).

It should further be noted that the breaking cycles method does not rely on any specific code construction algorithm. In this section the PEG construction was merely used as an example since it is one of the most prominent code construction methods. Even though, for this specific combination of code construction and dummy bit insertion, it might be possible to incorporate the cycle breaking approach directly into the process of constructing the PEG code (which is also based on the idea of avoiding short cycles), here, both processes are considered as separate entities for the sake of generality.

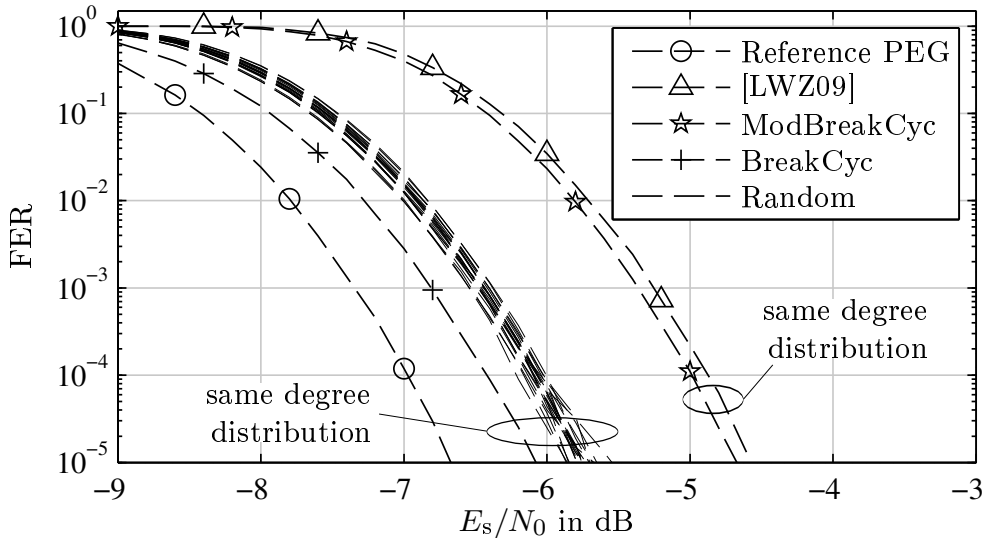


Figure 3.5: Influence of different degree distributions on the breaking cycles Algorithm 3 (BreakCyc) and [LWZ09] for shortened code rate $r_s = \frac{1}{10}$.

3.2 Parity Puncturing

The previous section describes a technique to flexibly reduce the code rate of a mother code with fixed code rate r . The method of *parity puncturing*, on the other hand, increases the code rate r_p , with $r < r_p \leq 1$, by simply discarding an arbitrary number of the parity bits generated by the mother code before transmission. Parity puncturing is a well-known method that was developed especially for convolutional codes at first [CCG79, Hag88]. In [LN02] the concept was extended to the case of regular LDPC codes and later also to irregular LDPC codes in [HM03a, HKM04].

In the following, the general concept of parity puncturing is introduced, followed by an asymptotic analysis on the ensemble performance and some finite length optimization techniques for selecting the exact bit positions to be punctured.

3.2.1 Encoding and Decoding⁶

In Figure 3.6 block diagrams of encoder and decoder using parity puncturing are shown. As in the case of information shortening, the core implementation of the mother encoder and decoder can be used without modifications and only the removal and insertion fixed values has to be executed additionally.

Encoding

The (N, K) mother code of rate r takes the information word $\mathbf{x}$ of length K as input and generates $M = \frac{1-r}{r}K$ parity bits at the output. Then, M_d of the parity bits are simply discarded prior to transmission. Thus, the transmitted code word $\mathbf{y}$ only consists of the K information bits and the remaining $M_p = M - M_d$ parity bits, resulting in an $(N - M_d, K)$ code with effective code length $N_p = K + M_p$ and effective code rate

$$r_p = \frac{K}{N_p} = \frac{K}{K + M_p} = \frac{\frac{r}{1-r}}{\frac{r}{1-r} + \frac{M_p}{M}} = \frac{1}{1 + \frac{1-r}{r}m_p}, \quad (3.20)$$

where the relative punctured parity word and code word lengths are defined as m_p and n_p by

$$m_p := \frac{M_p}{M} = \frac{r_p^{-1} - 1}{r^{-1} - 1}, \quad (3.21)$$

and

$$n_p := \frac{N_p}{N} = \frac{r}{r_p}. \quad (3.22)$$

Any code rate $r \leq r_p \leq 1$ can be achieved by an accordant choice of m_p (again only in discrete steps, depending on the absolute values of M_p and M). If no bits are punctured (i.e., $m_p = 1$), $r_p = r$ follows and if all M parity bits are punctured (i.e., $m_p = 0$), the effective code rate is $r_p = 1$.

⁶The description in this section follows the pre-published paper [BV14].

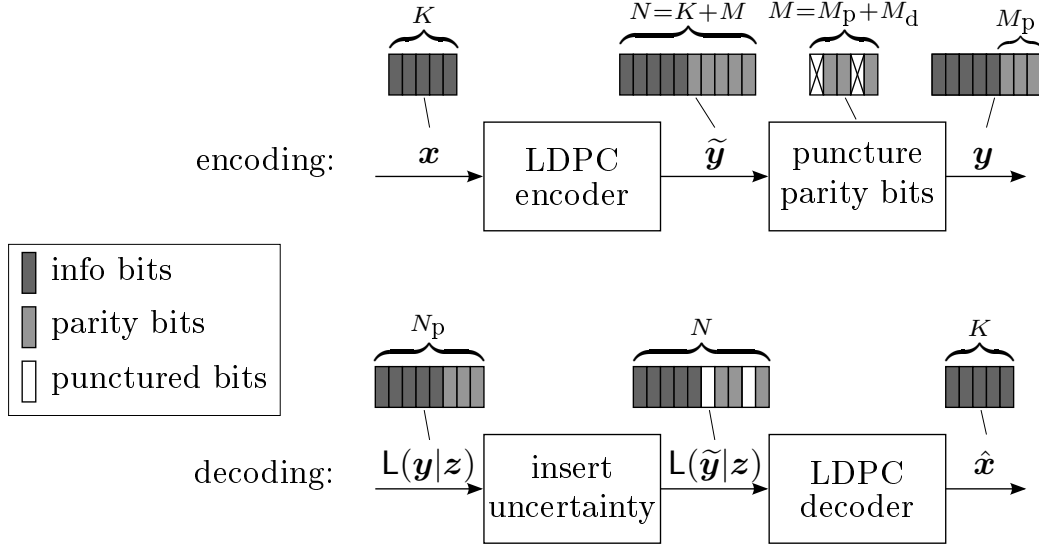


Figure 3.6: Encoding and decoding using a fixed mother encoder and decoder and parity puncturing to adapt the code rate. The different colors visualize exemplary positions of information, parity, and punctured bits.

Decoding

At the receiver, no channel information about the punctured bits is available and, thus, zeros are inserted into the received LLR sequence $L(y|z)$ according to the punctured positions, yielding the LLR sequence $L(\tilde{y}|z)$. LLRs with value zero correspond to complete uncertainty as the binary values 0/1 are equally likely. For the computation of messages according to the BP check node update, the messages with complete uncertainty are dominant in the sense of $L_i \boxplus 0 = 0$ (see, e.g., [HOP96]). Due to this dominance of uncertainty, punctured nodes first have to be *recovered* by neighboring unpunctured nodes, before a connected check node can pass non-zero messages to other variable nodes. This behavior degrades the decoding performance and has to be considered in the design of efficient puncturing patterns. As in the case of information shortening, existing decoder implementations of the mother code do not have to be modified.

In analogy to information shortening, encoding and decoding with arbitrary positions of punctured bits can be equivalently realized by simply reordering the check matrix columns such that the punctured positions are in the last M_d positions.

3.2.2 Asymptotic Analysis and Optimization

While shortened positions are described by a degree distribution (with coefficients summing up to one) in the previous section, a different representation is commonly chosen to describe the punctured positions. In accordance with [HKM04], the *puncturing polynomial* of effective rate r_p is defined by

$$\Pi^{[r_p]}(x) = \sum_{i=2}^{d_v} \Pi_i^{[r_p]} x^i, \quad (3.23)$$

where $0 \leq \Pi_i^{[r_p]} \leq 1$ denotes the fraction of punctured degree- i variable nodes. More precisely, the absolute number of punctured variable nodes of degree i is given by

$$M_{d,i} := \Pi_i^{[r_p]} \Lambda_i N = \tilde{\Pi}_i^{[r_p]} N. \quad (3.24)$$

It should be noted that the coefficients $\Pi_i^{[r_p]}$ do not sum up to one (as in the case of a degree distribution), but instead are related to the mother code by

$$\begin{aligned} \tilde{\Pi}^{[r_p]}(x) \Big|_{x=1} &= \sum_{i=2}^{d_v} \tilde{\Pi}_i^{[r_p]} = \sum_{i=2}^{d_v} \Pi_i^{[r_p]} \Lambda_i \\ &= \sum_{i=2}^{d_v} \frac{M_{d,i}}{N} = \frac{M_d}{N} = 1 - n_p \stackrel{(3.22)}{=} 1 - \frac{r}{r_p}. \end{aligned} \quad (3.25)$$

Threshold Computation using Gaussian Approximation

The asymptotic effect of punctured bits on the ensemble threshold was incorporated into density evolution using a Gaussian approximation (cf. Section 2.7.1) in [HM03a, HM03b, HKM04] and is briefly reviewed in the following. The ensemble threshold of a punctured code of effective rate r_p is determined by the triple $(\lambda, \rho, \Pi^{[r_p]})$, i.e., by the degree distribution pair of the mother code and the puncturing polynomial, specifying how many nodes of each degree are punctured (where a random selection of the punctured nodes is assumed for each degree in the following ensemble threshold analysis). Firstly, let $e^{(k)} := \mathbb{P}(\text{"}v_{n,m}^{(k)} = 0 \text{ in } k\text{-th iteration"})$ denote the probability of an arbitrary variable-to-check message $v_{n,m}^{(k)}$ being equal to zero in the k -th iteration due to puncturing and likewise $\varepsilon^{(k)} := \mathbb{P}(\text{"}c_{m,n}^{(k)} = 0 \text{ in } k\text{-th iteration"})$ the probability of an arbitrary check-to-variable message $c_{m,n}^{(k)}$ being equal to zero in the k -th iteration due to puncturing. As described above, a check-to-variable message is zero if at least one of the incoming variable-to-check messages that are summed up (using the boxplus operation) is equal to zero. Thus, the probability $\varepsilon^{(k)}$ can be written as

$$\begin{aligned} \varepsilon^{(k)} &= \sum_{j=2}^{d_c} \underbrace{\mathbb{P}\left(\text{"edge connected to degree } j \text{ check node"}\right)}_{=\rho_j} \mathbb{P}\left(\text{"at least one of } j-1 \text{ incoming messages is zero"}\right) \\ &= \sum_{j=2}^{d_c} \rho_j \underbrace{\left(1 - \underbrace{\left(\underbrace{1 - e^{(k)}}_{\text{specific message } \neq 0}\right)^{j-1}}_{\text{all } j-1 \text{ messages } \neq 0}\right)}_{\text{at least one message } = 0}. \end{aligned} \quad (3.26)$$

A variable-to-check message, on the other hand, is only equal to zero if the channel LLR is zero, i.e., the respective variable node is punctured, and all incoming check-

to-variable messages are zero.⁷ Thus, the probability $e^{(k)}$ is given by

$$\begin{aligned}
 e^{(k)} &= \sum_{i=2}^{d_v} \underbrace{\mathrm{P}\left(\begin{array}{c} \text{"edge connected to"} \\ \text{deg. } i \text{ variable node"} \end{array}\right)}_{=\lambda_i} \underbrace{\mathrm{P}\left(\begin{array}{c} \text{"degree } i \text{ variable"} \\ \text{node is punctured"} \end{array}\right)}_{=\Pi_i^{[rp]}} \mathrm{P}\left(\begin{array}{c} \text{"all incoming"} \\ \text{messages zero"} \end{array}\right) \\
 &= \sum_{i=2}^{d_v} \lambda_i \Pi_i^{[rp]} \underbrace{\left(\underbrace{\varepsilon^{(k-1)}}_{\text{specific message} = 0} \right)^{i-1}}_{\text{all } i-1 \text{ messages} = 0}. \tag{3.27}
 \end{aligned}$$

By inserting (3.26) into (3.27), the recursion

$$e^{(k)} = \sum_{i=2}^{d_v} \lambda_i \Pi_i^{[rp]} \left(\sum_{j=2}^{d_c} \rho_j \left(1 - \left(1 - e^{(k-1)} \right)^{j-1} \right) \right)^{i-1}, \tag{3.28}$$

is obtained for $k \geq 0$, where the initial value is set to $e^{(-1)} = 1$. Under the Gaussian assumption [CRU01], the message probability densities can be described by a mixture of a continuous Gaussian density and a discrete point mass (i.e., a Dirac impulse) at zero corresponding to the current probability of a message to be equal to zero. Similarly to the unpunctured case (described in Section 2.7.2) a recursion for the mean $\mu_c^{[k]}$ of the continuous part of the density can be derived which takes into account the probabilities $e^{(k)}$ and $\varepsilon^{(k)}$ of erased messages [HKM04]:

$$\begin{aligned}
 \mu_c^{[k]} &= \sum_{j=2}^{d_c} \rho_j \phi^{-1} \left(1 - \frac{1}{(1 - e^{(k)})^{j-1}} \left(1 - \sum_{i=2}^{d_v} \left(\lambda_i \Pi_i^{[rp]} \sum_{l=0}^{i-1} \chi_{i-1,l}^{(k-1)} \phi(l\mu_c^{[k-1]}) \right. \right. \right. \\
 &\quad \left. \left. \left. + \lambda_i (1 - \Pi_i^{[rp]}) \sum_{l=0}^{i-1} \chi_{i-1,l}^{(k-1)} \phi(l\mu_c^{[k-1]} + \mu_L) \right) \right)^{j-1} \right). \tag{3.29}
 \end{aligned}$$

Where $\phi(\cdot)$ was introduced in (2.66) on page 28 and the auxiliary term $\chi_{i-1,l}^{(k-1)}$ describes the binomial distribution

$$\chi_{i-1,l}^{(k-1)} = \binom{i-1}{l} \left(\varepsilon^{(k-1)} \right)^{i-1-l} \left(1 - \varepsilon^{(k-1)} \right)^l. \tag{3.30}$$

Furthermore, it was shown in [HKM04] that the probability of error for the information bits can only approach zero if also all (punctured) parity bits are successfully recovered, i.e., if $e^{(k)} \rightarrow 0$ and $\varepsilon^{(k)} \rightarrow 0$ for $k \rightarrow \infty$. Using the same methods as described in Section 2.7.1 for the unpunctured case, the *punctured ensemble threshold* $\tau_{\text{GADE}}(\lambda, \rho, \Pi^{[rp]})$ can be computed using recursion (3.29). However, due to the Gaussian approximation, the precision of the predicted ensemble thresholds strongly depends on the code rate and the maximum variable node degree.

⁷With fixed point arithmetic, a message can also be equal to zero if multiple non-zero messages are added. This case, however, does not need to be considered, since here only the zero-messages that are caused by puncturing are of interest.

Threshold Computation using DDE

To achieve higher precision of the punctured ensemble threshold prediction, especially for code rates below 0.5 and/or large maximum variable node degrees, the effect of puncturing can also be incorporated into the DDE method (cf. Section 2.7.1). Combining the analysis of puncturing with DDE was briefly mentioned in [HKM04] but not specifically described in detail. Thus, in the following, a description of how to integrate puncturing into the DDE analysis is provided.

With the same notation as introduced in Section 2.7.1 for DDE and recursions (3.26) and (3.27) for $e^{(k)}$ and $\varepsilon^{(k)}$, puncturing can be incorporated into the DDE method as follows. The probability mass function $P_v^{[k]}(x = c\Delta x)$ of a variable-to-check message after the k -th iteration is decomposed into two parts:

$$P_v^{[k]}(x) = P_{v,0}^{[k]}(x) + P_{v,\neq 0}^{[k]}(x) = e^{(k)}\delta(x) + \left(1 - e^{(k)}\right) P_{v,\neq 0}^{[k]}(x), \quad (3.31)$$

where $P_{v,0}^{[k]}(x) = e^{(k)}\delta(x)$ describes the point mass at zero that is induced by the puncturing of variable nodes, and $P_{v,\neq 0}^{[k]}(x)$ describes the contribution of unpunctured or already recovered messages. The evolution of $P_{v,0}^{[k]}(x)$ is fully determined by recursion (3.27), while $P_{v,\neq 0}^{[k]}(x)$ is computed by the modified variable node update

$$\begin{aligned} P_{v,\neq 0}^{[k]} = & \sum_{i=2}^{d_v} \lambda_i \Pi_i^{[rp]} \sum_{l=1}^{i-1} \chi_{i-1,l}^{(k)} \left(P_{c|\neq 0}^{[k]}\right)^{\otimes l} \\ & + \lambda_i \left(1 - \Pi_i^{[rp]}\right) \sum_{l=0}^{i-1} \chi_{i-1,l}^{(k)} \left(P_{c|\neq 0}^{[k]}\right)^{\otimes l} \otimes \mathbf{p}_{\text{AWGN,LLR}}, \end{aligned} \quad (3.32)$$

summing over only those probabilities with contributions of at least one unpunctured term. The auxiliary term $\chi_{i,l}^{(k)}$ is given by (3.30). The same decomposition is done for the pmf $P_c^{[k]}(x = c\Delta x)$ of a check-to-variable message after the k -th iteration:

$$P_c^{[k]}(x) = P_{c,0}^{[k]}(x) + P_{c,\neq 0}^{[k]}(x) = \varepsilon^{(k)}\delta(x) + \left(1 - \varepsilon^{(k)}\right) P_{c,\neq 0}^{[k]}(x), \quad (3.33)$$

where $P_{c,0}^{[k]}(x) = \varepsilon^{(k)}\delta(x)$ describes the point mass at zero that represents zero check-to-variable messages of check nodes that are connected to punctured variable nodes and $P_{c,\neq 0}^{[k]}(x)$ describes the contribution of check nodes connected to unpunctured or already recovered variable nodes (and $P_{c|\neq 0}^{[k]}(x)$ describes the respective conditional pdf). Then $P_{c,0}^{[k]}(x)$ is fully determined by recursion (3.26), while $P_{c|\neq 0}^{[k]}(x)$ can be

computed by the modified check node update

$$\begin{aligned}
P_{c| \neq 0}^{[k]}(c\Delta x) &= \sum_{j=2}^{d_c} \rho_j \sum_{\substack{(c_1, \dots, c_{j-1}): \\ Q(c_1 \Delta x \boxplus \dots \boxplus c_{j-1} \Delta x) = c\Delta x}} \prod_{j'=1}^{j-1} P_{v| \neq 0}^{[k-1]}(c_{j'} \Delta x) \\
&= \sum_{j=2}^{d_c} \rho_j \sum_{\substack{(c_1, \dots, c_{j-1}): \\ Q(c_1 \Delta x \boxplus \dots \boxplus c_{j-1} \Delta x) = c\Delta x}} \prod_{j'=1}^{j-1} \frac{P_{v, \neq 0}^{[k-1]}(c_{j'} \Delta x)}{1 - e^{(k-1)}} \\
&= \sum_{j=2}^{d_c} \frac{\rho_j}{(1 - e^{(k-1)})^{j-1}} \sum_{\substack{(c_1, \dots, c_{j-1}): \\ Q(c_1 \Delta x \boxplus \dots \boxplus c_{j-1} \Delta x) = c\Delta x}} \prod_{j'=1}^{j-1} P_{v, \neq 0}^{[k-1]}(c_{j'} \Delta x),
\end{aligned} \tag{3.34}$$

for $c = \lceil \frac{x_{\min}}{\Delta x} - \frac{1}{2} \rceil, \dots, \lfloor \frac{x_{\max}}{\Delta x} + \frac{1}{2} \rfloor$. Equations (3.32) and (3.34) together with (3.26) and (3.27) are evaluated recursively, starting with $e^{(-1)} = 1$, $P_{c| \neq 0}^{[0]}(x) = \delta(x)$, and $P_{v| \neq 0}^{[0]} = P_{\text{AWGN,LLR}}$, to evaluate if error-free decoding is asymptotically possible at a given channel quality with a code from the punctured ensemble $(\lambda, \rho, \Pi^{[r_p]})$. By evaluating convergence at multiple channel qualities as described in Section 2.7.1, the punctured ensemble threshold $\tau_{\text{DDE}}(\lambda, \rho, \Pi^{[r_p]})$ can be computed.

Constrained Optimization

Given a fixed mother code degree distribution pair (λ, ρ) with code rate r , the asymptotic performance of a punctured code of code rate r_p can be optimized by minimizing the punctured ensemble threshold of the triple $(\lambda, \rho, \Pi^{[r_p]})$, where λ and ρ are fixed and the coefficients of $\Pi^{[r_p]}$ are the optimization variables. Under the assumption that the mother code degree distribution pair (λ, ρ) fulfills the stability condition (2.50), the punctured ensemble automatically also fulfills the stability condition since

$$\lambda_2 \stackrel{(2.50)}{<} \frac{e^{r\tau(\lambda, \rho)}}{\sum_{j=2}^{d_c} \rho_j (j-1)} < \frac{e^{r_p \tau(\lambda, \rho, \Pi^{[r_p]})}}{\sum_{j=2}^{d_c} \rho_j (j-1)} \tag{3.35}$$

directly follows from (2.50) and $r_p > r$ as well as $\tau(\lambda, \rho, \Pi^{[r_p]}) \geq \tau(\lambda, \rho)$, where the latter inequality holds because the ensemble threshold cannot be reduced (i.e., improved) by discarding the information of punctured bits. Instead, a similar condition is proposed here for the puncturing fraction $\Pi_2^{[r_p]}$ as a necessary condition for the residual puncturing probability $e^{(k)}$ (or equivalently $\varepsilon^{(k)}$) to converge to zero. For this purpose, the first-order Taylor series approximation of the recursion (3.28) is examined in the following. Firstly, (3.28) is considered as describing $e^{(k)}$ as a function of $e^{(k-1)}$:

$$e^{(k)} = f(e^{(k-1)}). \tag{3.36}$$

The derivative of $f(e^{(k-1)})$ with respect to $e^{(k-1)}$ is denoted $f'(e^{(k-1)})$. To evaluate the behavior of this function close to the point zero, the first-order Taylor series is determined around $e^{(k-1)} = 0$ by

$$\begin{aligned} e^{(k)} = f(e^{(k-1)}) &\approx \underbrace{f(e^{(k-1)} = 0)}_{=0} + \underbrace{\frac{f'(e^{(k-1)} = 0)}{1!}}_{=\lambda_2 \Pi_2^{[rp]} \rho'(1)} \cdot (e^{(k-1)} - 0) \\ &= e^{(k-1)} \lambda_2 \Pi_2^{[rp]} \rho'(1), \end{aligned} \quad (3.37)$$

with $\rho'(1) = \frac{d\rho(x)}{dx}|_{x=1}$. The value $f'(e^{(k-1)} = 0)$ can be obtained by derivating (3.28). Altogether, for $e^{(k)}$ to converge to zero, the linear factor in recursion (3.37) needs to be less than 1:

$$\begin{aligned} \frac{e^{(k)}}{e^{(k-1)}} &\approx \lambda_2 \Pi_2^{[rp]} \rho'(1) \stackrel{!}{<} 1 \\ \Leftrightarrow \quad \Pi_2^{[rp]} &\stackrel{!}{<} \frac{1}{\lambda_2 \rho'(1)} = \frac{1}{\lambda_2 \sum_{j=2}^{d_c} \rho_j(j-1)}. \end{aligned} \quad (3.38)$$

This inequality will be referred to as *puncturing stability condition*. Similarly to the stability condition (2.50), it can be used as additional constraint during the optimization of a puncturing polynomial as it is a necessary condition for convergence to zero error probability.

Assuming stability of the mother code ensemble, the optimization problem of finding a good puncturing polynomial is then given by

$$\min \quad \tau(\lambda, \rho, \Pi^{[rp]}) \quad (3.39a)$$

$$\text{subject to} \quad \sum_{i=2}^{d_v} \Pi_i^{[rp]} \Lambda_i \stackrel{(3.25)}{=} 1 - \frac{r}{r_p} \quad (3.39b)$$

$$0 \leq \Pi_i^{[rp]} \leq 1, \quad i = 2, \dots, d_v \quad (3.39c)$$

$$\Pi_2^{[rp]} \stackrel{(3.38)}{<} \frac{1}{\lambda_2 \rho'(1)}. \quad (3.39d)$$

This problem can be solved using global optimization with DDE or a linearization based on the GA (details on the linearization are found in [HKM04]).

3.2.3 Finite Length Optimization: Guaranteed Recoverability

As in the case of information shortening, finite length effects caused by parity puncturing describe effects that are not related to the punctured ensemble performance

but to the concrete structure of a punctured code realization. A number of detrimental effects of punctured nodes in the Tanner graph and matrix construction algorithms taking these effects into account have been investigated in the literature, e.g., [HKKM06, ZMGC12, AB13]. The most important and widely used property of punctured nodes is the so-called *k-step-recoverability* [HKKM06]. Before decoding starts, no intrinsic information about a punctured parity bit is available from the channel observation. Thus, the value of the parity bit has to be *recovered* during iterative decoding by collecting extrinsic information from the check nodes connected to the punctured variable node. A punctured node is said to be *k-step-recoverable* (*k*-SR) if it receives at least one non-zero message for the first time in the *k*-th iteration, assuming that a flooding schedule is used.⁸ It can roughly be assumed that the earlier a punctured node is recovered, the more reliable its recovery is. Following this assumption, a typical finite length optimization consists in selecting the punctured variable nodes such that as many nodes as possible with low *k*-step-recoverability are created. Examples of such algorithms have been presented in [HKKM06] and more recently in [AB13].

Strongly Punctured Codes with Guaranteed Recoverability⁹

The more parity bits are punctured, the more likely it becomes to create *unrecoverable* punctured nodes, also called ∞ -step-recoverable (∞ -SR). This usually happens very suddenly at a specific punctured code rate, and as soon as one node becomes unrecoverable, almost all other punctured nodes will also be unrecoverable due to mutual dependencies. Once this happens, the error correction performance of the punctured code becomes very poor, as nearly the whole graph is deactivated by the unrecoverable nodes no matter how many iterations are executed.

A low-complexity approach that, during construction of a check matrix, effectively avoids the creation of unrecoverable nodes even at very high punctured rates is presented. This approach has also been employed in the code construction method introduced in [BWV14]. The presence of punctured nodes in the Tanner graph, which have to be recovered before they can forward extrinsic information, unavoidably induces a performance loss compared to a dedicated unpunctured code of the same rate. While this penalty is relatively small for moderate fractions of punctured nodes, it strongly increases for large puncturing fractions. In contrast to most previous approaches, the proposed method does not aim at optimizing the performance at low to moderate puncturing fractions, but instead enables puncturing up to an arbitrarily high punctured code rate without creating unrecoverable nodes. This is achieved by modifying the underlying code construction method (e.g., the PEG algorithm) as follows.

Before starting the construction of the parity check matrix, the desired number M_d of the variable nodes are marked as punctured nodes. Each of these nodes needs to

⁸It should be noted that this definition of the recovery of a punctured node does not account for whether a node is actually recovered correctly, but simply measures the number of iterations that have to be executed until *any* non-zero message arrives at that particular node.

⁹The description of this algorithm follows the description given in the pre-published paper [BWV14].

be made sure to be recoverable at the end of the matrix construction process. To achieve this, for each punctured variable one dedicated *guaranteed survived check node* [HKKM06] is reserved, which is guaranteed to be capable of recovering the connected punctured node. Thus, M_d (arbitrary) check nodes are initially marked as guaranteed survived check node and the remaining $M_p = M - M_d$ check nodes are declared as *potentially dead check node* [HKKM06], as they are not known to have any guaranteed properties. Whenever processing one of the punctured variable nodes during code construction, it is first connected to exactly one of the reserved guaranteed survived check nodes. Since none of these check nodes has been connected to any other punctured node so far, the current punctured node will be 1-SR as it can be directly recovered by the guaranteed survived check node within one iteration. Before proceeding, the connected guaranteed survived check node is marked as potentially dead check node, as recoverability cannot be guaranteed by this check node if another punctured node is also connected to it. Since, however, this check node might later be connected to other punctured nodes, not all punctured nodes will remain 1-SR. For the remaining connections of the current variable nodes no further guaranteed survived check node but only potentially dead check nodes can be chosen. As long as the underlying code construction algorithm exhibits a certain randomness, creating long chains of dependent punctured nodes with the proposed method is very unlikely. Empirical observations show that if the proposed method is combined with the PEG algorithm, nearly all (up to 98% in the example below) of the parity bits can be punctured and still be recoverable within a few iterations. An intermediate step during code construction with the proposed modification is shown in Figure 3.7.

Simulation Results

To evaluate the proposed method of guaranteed recoverability, a rate-1/2 LDPC mother code of length $N = 2394$ is constructed with the PEG algorithm and the optimized degree distribution $\lambda(x)$ from Table 3.4. The SA optimizer described

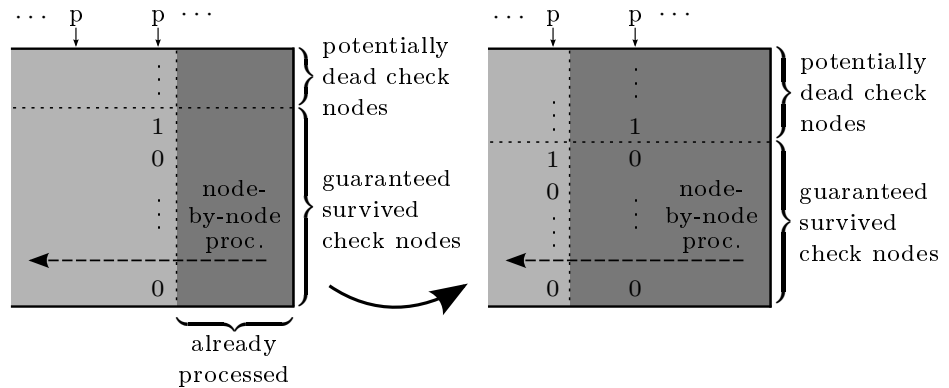


Figure 3.7: Achieving guaranteed recoverability by connecting every punctured node (columns marked by “p”) to exactly one *guaranteed survived check node* (i.e., placing a one in the upper right corner of the light gray shaded area) during code construction with node-by-node processing of the check matrix.

r	Λ_2	Λ_3	Λ_4	Λ_5	Λ_{10}	$\tau_{\text{DDE}}(\Lambda, R) _{\text{dB}}$
0.5	0.490	0.320	0.010	0.035	0.145	-2.631
r_p	$\Pi_2^{[r_p]}$	$\Pi_3^{[r_p]}$	$\Pi_4^{[r_p]}$	$\Pi_5^{[r_p]}$	$\Pi_{10}^{[r_p]}$	$\tau_{\text{DDE}}(\Lambda, R, \Pi^{[r_p]}) _{\text{dB}}$
0.6	0.205	0.084	0.000	0.293	0.202	-1.305
0.7	0.349	0.176	0.000	0.486	0.285	-0.012
0.8	0.435	0.307	0.000	0.594	0.297	1.378
0.9	0.531	0.346	0.000	0.824	0.306	3.594
0.95	0.518	0.463	0.407	0.176	0.423	4.937
0.98	0.534	0.479	0.423	0.192	0.439	$e^{(k)} \nrightarrow 0$

Table 3.4: Mother code degree distribution $\lambda(x)$ for $r = 0.5$ and puncturing polynomials $\Pi^{[r_p]}(x)$ for punctured code rates $r_p \in \{0.6, 0.7, 0.8, 0.9, 0.95, 0.98\}$ optimized with SA and DDE with 8 quantization bits.

in Section 2.7.2 with DDE and 8 bit quantization is used to optimize puncturing polynomials for punctured code rates $r_p \in \{0.6, 0.7, 0.8, 0.9, 0.95, 0.98\}$ according to the optimization problem (3.39). The results are given in Table 3.4. For $r_p = 0.98$ no puncturing polynomial could be found for which the residual puncturing fraction $e^{(k)}$ (and thus the error probability) converges to zero. For all other code rates, the DDE thresholds τ_{DDE} in terms of E_s/N_0 values are given in the table.

The following three setups of puncturing are compared.

Random : $\Pi_i^{[r_p]} \Lambda_i N$ punctured nodes are selected randomly from all degree- i variable nodes at each punctured code rate r_p and each degree i .

Proposed guaranteed recoverability : during PEG construction, for each punctured code rate r_p and for each degree i , the according number $M_{d,i} = \Pi_i^{[r_p]} \Lambda_i N$ of punctured nodes are made sure to be connected to exactly one still guaranteed survived check node. The remaining connections of the punctured variable nodes are then chosen only from the potentially dead check nodes by the PEG algorithm.

Reference [HKKM06]: Punctured nodes are selected according to the intentional reference puncturing scheme based on *grouping and sorting* (and irrespective of the puncturing polynomials from Table 3.4).

In Figure 3.8 histograms of the k -step-recoverability of punctured nodes are shown for the three described setups and all punctured rates r_p . At code rates $r_p = 0.6$ and $r_p = 0.7$ the random selection and the proposed approach show very similar distributions of k -step-recoverability between $k = 1$ and $k = 5$, while the reference approach [HKKM06] produces only 1-SR nodes and very few additional 2-SR nodes at $r_p = 0.7$. At $r_p = 0.8$, the reference scheme still produces the lowest average recoverability level $\bar{k}$ but the proposed scheme already achieves better recoverability than the random scheme. The situation changes at $r_p = 0.9$, where the random scheme already suffers from unrecoverable nodes and also the reference scheme starts to exhibit large

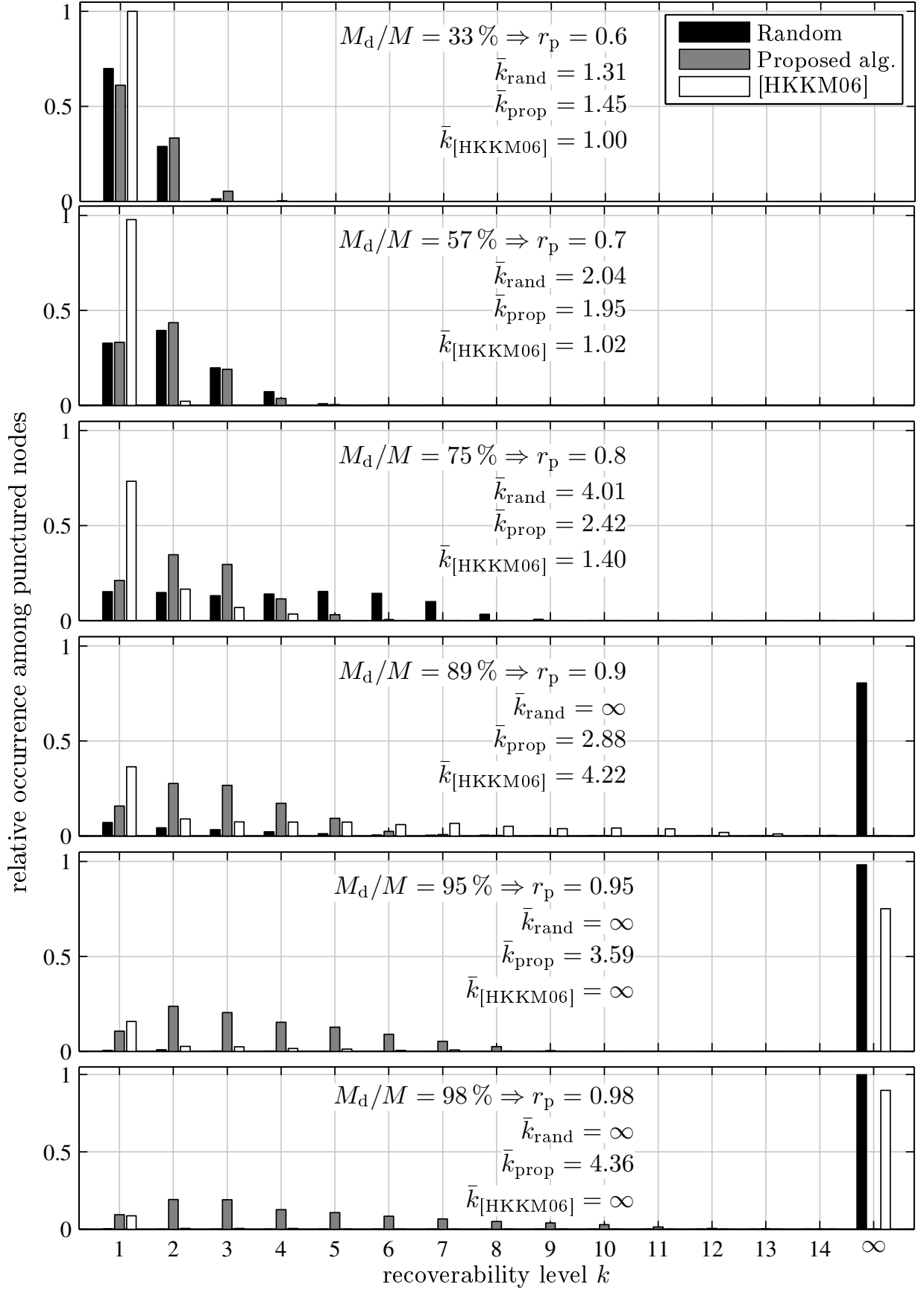


Figure 3.8: Histograms of k -step-recoverability of three puncturing schemes.

recoverability levels of up to 14. While it still manages to create the largest number of 1-SR nodes, the reference scheme is outperformed by the proposed scheme in terms of the average recoverability $\bar{k}$. Finally, at $r_p = 0.95$ and $r_p = 0.98$ both the random and the reference scheme create unrecoverable nodes, while the proposed scheme effectively avoids these and at the same time manages to keep the average recoverability level $\bar{k}_{\text{prop}}$ as low as 4.36 for $r_p = 0.98$. This is especially noteworthy since for this punctured rate almost all (98%) of the parity bits are punctured, and also the density evolution analysis predicted the presence of unrecoverable nodes (since for the employed puncturing polynomial $e^{(k)} \rightarrow 0$).

The behavior observed for the recoverability levels of the three considered setups is confirmed by the BER simulations with $I_{\text{max}} = 100$ iterations of BP decoding shown in Figure 3.9. The reference scheme [HKKM06] shows best performance for code rates $r_p \in \{0.6, 0.7, 0.8\}$ for which it also exhibits the lowest average recoverability level. For code rates $r_p \in \{0.9, 0.95, 0.98\}$, however, the proposed scheme shows the best performance, as expected from the recoverability histograms. The random scheme shows the worst performance for all punctured code rates. It can also be seen, that all cases at which unrecoverable nodes are present (i.e., $r_p \in \{0.9, 0.95, 0.98\}$ for the random scheme and $r_p \in \{0.95, 0.98\}$ for the reference scheme) show the same poor performance, which also corresponds to the BER curve of uncoded transmission.

Altogether, the proposed guaranteed recoverability scheme is a very low-complexity extension, suitable for many code construction methods that connect variable nodes in a node-by-node manner. While for moderate puncturing fractions more sophisticated methods provide better performance, the main strength of the proposed scheme is that an arbitrarily large fraction of the parity nodes can be punctured (resulting in code rates arbitrarily close to 1) without creating any unrecoverable nodes.

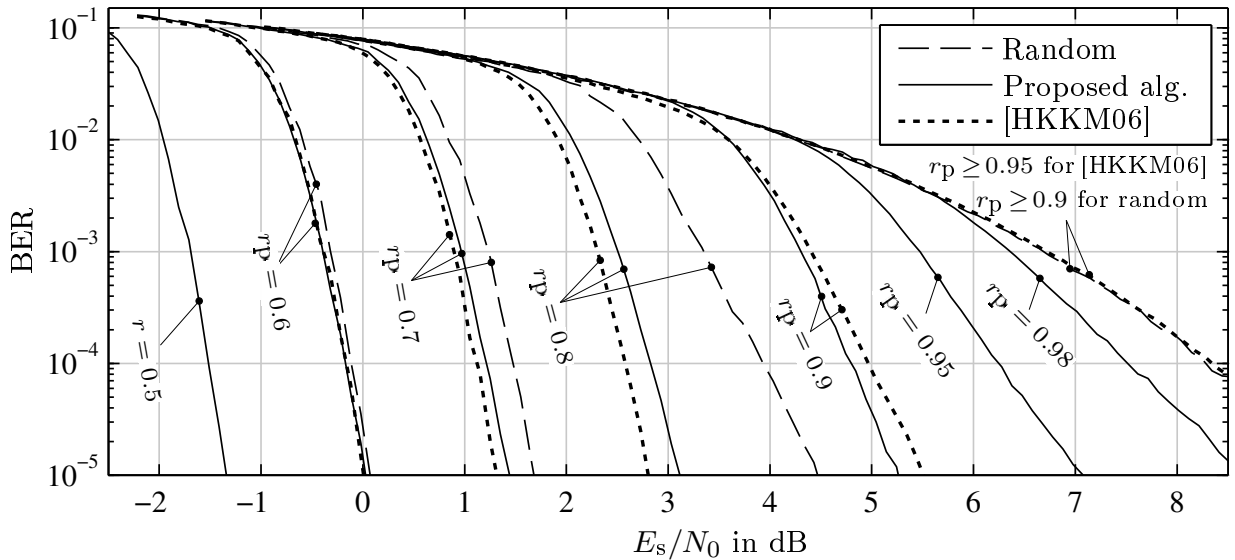


Figure 3.9: BER results for BP decoding with $I_{\text{max}} = 100$ iterations for the mother code of rate 1/2 and three different puncturing schemes achieving punctured code rates $r_p \in \{0.6, 0.7, 0.8, 0.9, 0.95, 0.98\}$.

3.3 Jointly Optimized Information Shortening and Parity Puncturing

For supporting very wide ranges of code rates with a single mother code, using only either information shortening or parity puncturing is not very practical. Both techniques have the drawback of decreasing the effective shortened or punctured code length to $n_s N = \frac{1-r}{1-r_s} N$ and $n_p N = \frac{r}{r_p}$, respectively. This reduced length causes a performance loss which is particularly distinctive if the mother code length N is already relatively small (e.g., below 1000). If the range $r_{\min} = 0.1$ to $r_{\max} = 0.9$ is covered by only either shortening or puncturing, the resulting effective code length will be reduced by up to almost one order of magnitude (to $\frac{N}{9}$). Furthermore, the additional penalty that is caused by punctured nodes quickly becomes dominant if the punctured code rate is far away from the mother code rate. This makes it virtually impossible to achieve code rate 0.9 by puncturing a mother code of code rate 0.1.

Much better performance can on average be achieved if shortening and puncturing are combined, using a single mother code of some intermediate code rate r , with $r_{\min} < r < r_{\max}$. Lower code rates r_s (with $r_{\min} \leq r_s < r$) can then be achieved by information shortening, while higher code rates r_p (with $r < r_p \leq r_{\max}$) are achieved by parity puncturing. This way, the disadvantages of having too many punctured nodes and a too heavily reduced code length are compensated to a great extent.

Figure 3.10 shows the effective relative code length $n_{\text{eff}} = n_s = \frac{N_s}{N}$ when shortening is used and $n_{\text{eff}} = n_p = \frac{N_p}{N}$ when puncturing is used over the effective code rate

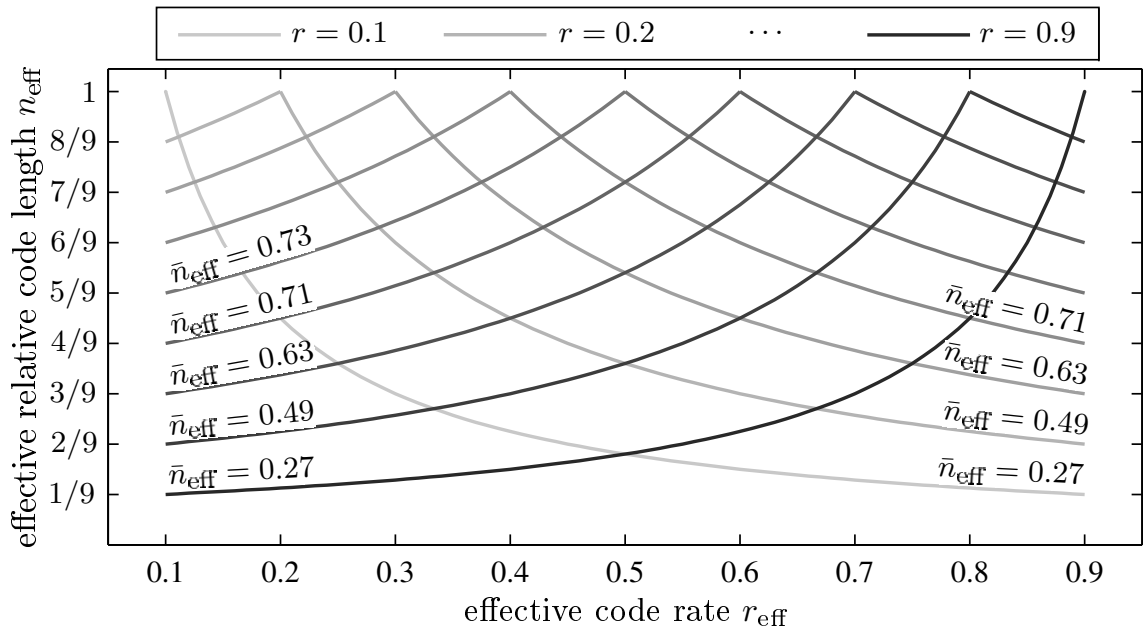


Figure 3.10: Effective relative code length n_{eff} when shortening is used to achieve code rates $r_{\text{eff}} < r$ and puncturing is used to achieve code rates $r_{\text{eff}} > r$ for different mother code rates from $r = 0.1$ (light) to $r = 0.9$ (dark). Each curve is labeled with its average relative code length $\bar{n}_{\text{eff}}$.

$r_{\text{eff}} = r_s$ or $r_{\text{eff}} = r_p$, respectively, for different mother code rates from $r = 0.1$ (light color) to $r = 0.9$ (dark color). Each curve is additionally labeled with its respective average relative effective code length, denoted $\bar{n}_{\text{eff}}$. To minimize the performance loss due to a decreased code length, the configuration with the maximum average code length should be chosen. In this case, the largest average code length is achieved if the mother code rate is chosen exactly in the middle of the supported code rate range. In general, however, this is not the case. Instead, under the assumption of equiprobable and densely spaced code rates in the range $r_{\min}$ to $r_{\max}$, it can be shown that the maximum average relative code length

$$\bar{n}_{\text{eff}}^* = \frac{\log(1 + r_{\max} - r_{\min})}{r_{\max} - r_{\min}} \quad (3.40)$$

is attained at the mother code rate

$$r^* = \frac{1}{1 + \frac{1-r_{\min}}{r_{\max}}}. \quad (3.41)$$

A derivation of this maximum relative code length is provided in Appendix A.

It should be noted, however, that being the optimal choice for the mother code rate in terms of maximizing the average relative code length $\bar{n}_{\text{eff}}$, does not imply that r^* is also optimal in terms of ensemble thresholds or error correction performance in general. This issue will be discussed in more detail below in Section 3.3.3.

In the following, a novel joint optimization strategy of shortening and puncturing is presented to realize a multi-rate LDPC code ensemble with one mother code of rate r . The presented method has been pre-published in [BV14]. A similar combination of shortening and puncturing was already used in [TJ05], where the authors successively optimized shortening parameters for a given fixed mother code with a given fixed puncturing scheme. In contrast, the novelty here lies in the joint optimization of all supported code rates, including the optimization of the mother code itself, as well as all parameters of information shortening and parity puncturing. For simplicity of encoding and decoding, shortening and puncturing is performed in a *rate-compatible* fashion, meaning that bits that are shortened at a code rate r_s also have to be shortened at all lower rates $r'_s < r_s$, and that bits that are punctured at a rate r_p also have to be punctured at all higher rates $r'_p > r_p$. Furthermore, for ease of notation, the columns of the mother code's check matrix are always ordered in such a way that shortening is performed column-by-column (i.e., bit-by-bit) starting from the lowest index and puncturing is performed column-by-column starting from the highest index. The code word structure of this rate-compatible scheme is depicted exemplarily for a mother code of rate $r = 1/2$ in Figure 3.11.

3.3.1 Asymptotic Analysis and Optimization¹⁰

As discussed in Sections 3.1.2 and 3.2.2, the asymptotic performance of shortening and puncturing depends on the shortened degree distribution pair and the puncturing polynomial, respectively. Given a sequence of ζ ascending target code rates

¹⁰The descriptions and results in this section follow the pre-published paper [BV14].

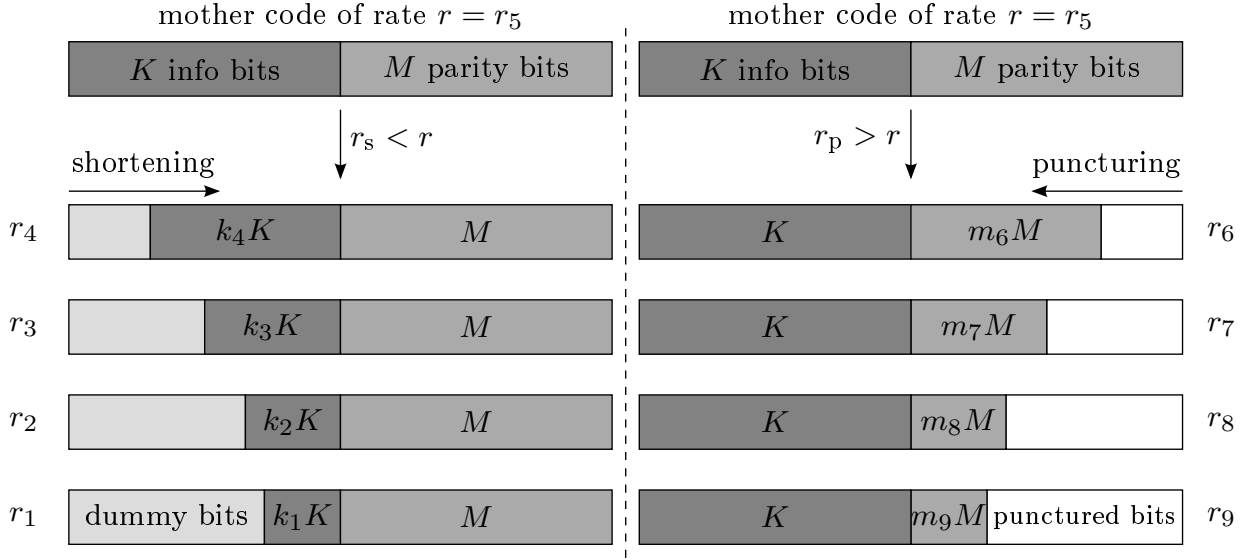


Figure 3.11: Exemplary rate-1/2 code word shortened to code rates $r_1, \dots, r_4$ (left) and punctured to code rates $r_6, \dots, r_9$ (right).

$r_1 < r_2 < \dots < r_\zeta$ in an arbitrary range between 0 and 1, the objective of the proposed joint rate-compatible optimization process is to find a mother code degree distribution pair of rate $r = r_\mu \in \{r_1, \dots, r_\zeta\}$ and corresponding shortened degree distribution pairs for the code rates below r as well as puncturing polynomials for the code rates above r , such that close to capacity performance is possible at all target rates. The output of the optimization consists of ζ polynomials:

- $\Lambda^{[r_\mu]}(x) = \Lambda(x)$: a variable node degree distribution for the mother code of rate $r = r_\mu \in \{r_1, \dots, r_\zeta\}$,
- $\Lambda^{[r_s]}(x)$: a shortened variable node degree distribution for each target code rate $r_s \in \{r_1, \dots, r_{\mu-1}\}$ with $r_s < r$,
- $\Pi^{[r_p]}(x)$: a puncturing polynomial for each target code rate $r_p \in \{r_{\mu+1}, \dots, r_\zeta\}$ with $r_p > r$.

Additionally, at all rates, the effective check node degree distributions are assumed to be concentrated (which is only valid if the check matrix is constructed accordingly, cf. Section 3.1.2).

Constraints

There are five constraints that are necessary for the joint optimization to yield a valid rate-compatible solution. The interrelations of the involved quantities are visualized in Figure 3.12.

1. Since shortening is performed in a rate-compatible way, the shortened columns

of each rate must also be included at the next lower rate (for all degrees i):

$$\begin{aligned} \Lambda_i N &\stackrel{(3.9)}{\geq} \Lambda_i^{[r_{\mu-1}]} n_{\mu-1} N \geq \dots \geq \Lambda_i^{[r_1]} n_1 N \\ \Leftrightarrow \frac{\Lambda_i}{1-r} &\geq \frac{\Lambda_i^{[r_{\mu-1}]}}{1-r_{\mu-1}} \geq \dots \geq \frac{\Lambda_i^{[r_1]}}{1-r_1}. \end{aligned} \quad (3.42)$$

This is equivalent to $\Delta_i^{[r_k]} \geq 0, \forall k$.

2. When shortening from one rate r_k to the next lower one r_{k-1} ($k \in \{2, \dots, \mu\}$), the number of shortened nodes of any degree i cannot exceed the total number of shortened nodes for that specific rate difference:

$$(1-r_{k-1}) \Lambda_i^{[r_k]} - (1-r_k) \Lambda_i^{[r_{k-1}]} \stackrel{(3.10)}{\leq} r_k - r_{k-1}. \quad (3.43)$$

This is equivalent to $\Delta_i^{[r_k]} \leq 1, \forall k$.

3. Punctured and shortened nodes must be disjunct, and thus the number of punctured nodes $M_{d,i} = \tilde{\Pi}_i^{[r_\zeta]} N$ of each degree i at the highest rate r_ζ must be less than or equal to the number $N_{s,i} = \Lambda_i^{[r_1]} n_1 N$ of degree- i nodes remaining in the degree distribution of the lowest shortened rate r_1 :

$$\tilde{\Pi}_i^{[r_\zeta]} = \Pi_i^{[r_\zeta]} \Lambda_i \leq n_1 \Lambda_i^{[r_1]} = \frac{1-r}{1-r_1} \Lambda_i^{[r_1]}. \quad (3.44)$$

4. Since puncturing is performed in a rate-compatible fashion, the punctured nodes of each rate must also be included at the next higher rate (for all degrees i):

$$\Pi_i^{[r_{\mu+1}]} \leq \Pi_i^{[r_{\mu+2}]} \leq \dots \leq \Pi_i^{[r_\zeta]}. \quad (3.45)$$

5. For each rate $r_p \in \{r_{\mu+1}, \dots, r_\zeta\}$ the puncturing polynomial $\Pi^{[r_p]}$ in conjunction with the mother code degree distribution $\Lambda(x)$ must result in the correct total number of punctured bits:

$$\tilde{\Pi}^{[r_p]}(1) = \sum_{i=2}^{d_v} \Pi_i^{[r_p]} \Lambda_i \stackrel{(3.25)}{=} 1 - \frac{r}{r_p}. \quad (3.46)$$

Cost Function

The (mother/shortened/punctured) ensemble thresholds τ_k are computed for each rate $r_k \in \{r_1, \dots, r_\zeta\}$ of the candidate solution by means of the methods described in Sections 3.1.2 and 3.2.2. The performance is assumed to be of equal importance at each target code rate r_k . Therefore, the *mean gap to capacity* is defined as a single

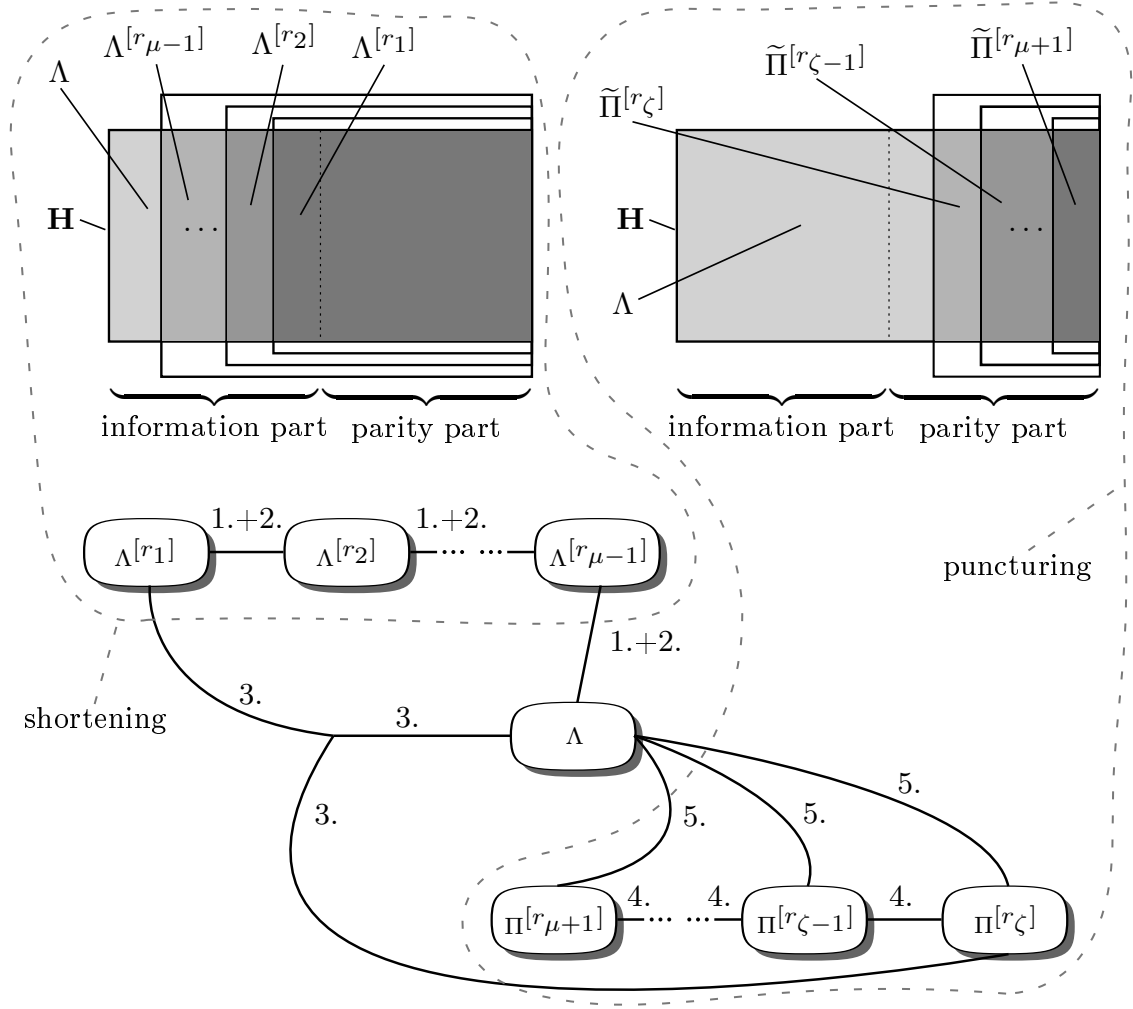


Figure 3.12: Visualization of relevant submatrices and rate-compatible constraints for the joint optimization of shortened degree distributions and puncturing polynomials.

value for a multi-rate ensemble with ζ code rates by

$$\begin{aligned}
 \bar{\gamma}|_{\text{dB}} &= \frac{1}{\zeta} \sum_{i=1}^{\zeta} \gamma_k|_{\text{dB}} = \frac{1}{\zeta} \sum_{i=1}^{\zeta} (\tau_k|_{\text{dB}} - \mathbf{q}(r_k)|_{\text{dB}}) \\
 &= \frac{1}{\zeta} \left(\underbrace{(\tau(\Lambda, R)|_{\text{dB}} - \mathbf{q}(r)|_{\text{dB}})}_{\text{mother code}} + \sum_{k=1}^{\mu-1} \underbrace{\left(\tau(\Lambda^{[r_k]}, R^{[r_k]})|_{\text{dB}} - \mathbf{q}(r_k)|_{\text{dB}} \right)}_{\text{shortened codes}} \right. \\
 &\quad \left. + \sum_{k=\mu+1}^{\zeta} \underbrace{\left(\tau(\Lambda, R, \Pi^{[r_k]})|_{\text{dB}} - \mathbf{q}(r_k)|_{\text{dB}} \right)}_{\text{punctured codes}} \right).
 \end{aligned} \tag{3.47}$$

in dB scale or equivalently by

$$\bar{\gamma} = \sqrt[\zeta]{\prod_{i=1}^{\zeta} \gamma_k} = \sqrt[\zeta]{\prod_{i=1}^{\zeta} \frac{\tau_k}{\mathbf{q}(r_k)}} \tag{3.48}$$

in linear scale. The value $\bar{\gamma}|_{\text{dB}}$ illustrates how many dB the multi-rate ensemble performs on average away from the channel capacity, while the linear value $\bar{\gamma}$ can be understood as the average factor in additional energy that is required for error free transmission compared to a set of perfectly capacity-achieving codes.

To ensure a certain “flatness” of a solution’s sequence of gaps $\gamma_k|_{\text{dB}}$, the cost function $C(\gamma_1, \dots, \gamma_\zeta)$ of the optimization process can alternatively be defined as the *mean square gap to capacity*

$$C(\gamma_1, \dots, \gamma_\zeta) = \frac{1}{\zeta} \sum_{k=1}^{\zeta} (\gamma_k|_{\text{dB}})^2. \quad (3.49)$$

While the mean of squared logarithmic values does not have a particularly illustrative meaning, its usage as cost function can be motivated as follows. By using squared values, the optimization at code rates with currently large gaps is rated more important than at those that already have close-to-capacity ensemble thresholds. E.g., if two ensemble thresholds are 2 dB and 1 dB away from capacity, improving the first one from 2 dB to 1.5 dB is preferred over improving the second one from 1 dB to 0.5 dB. Since rates are jointly optimized, it is even possible to accept a degradation at one rate to enable an improvement at other rates, which might be necessary due to constraints between the rates. In the following the mean square gap from (3.49) is used as cost function.

Additionally, it is possible to incorporate unequal importance levels of target code rates (e.g., due to more or less preferred operating points of a transmission system) into the optimization by assigning an appropriate weight w_k to each code rate and using an accordingly weighted mean distance or mean square distance

$$\frac{1}{\zeta} \sum_{k=1}^{\zeta} w_k \gamma_k|_{\text{dB}} \quad \text{or} \quad \frac{1}{\zeta} \sum_{k=1}^{\zeta} w_k \gamma_k|_{\text{dB}}^2 \quad (3.50)$$

as cost function.

Optimization Problem

Using the cost function and constraints described above, the joint multi-rate optimization problem is stated as

$$\min \frac{1}{\zeta} \sum_{k=1}^{\zeta} (\tau_k|_{\text{dB}} - \mathbf{q}(r_k)|_{\text{dB}})^2 \quad (3.51a)$$

$$\text{subject to } \frac{\Lambda_i^{[r_k]}}{1 - r_k} \stackrel{(3.42)}{\geq} \frac{\Lambda_i^{[r_{k-1}]}}{1 - r_{k-1}}, \quad \forall k \in \{2, \dots, \mu\} \quad (3.51b)$$

$$(1 - r_{k-1}) \Lambda_i^{[r_k]} - (1 - r_k) \Lambda_i^{[r_{k-1}]} \stackrel{(3.43)}{\leq} r_k - r_{k-1}, \quad \forall k \in \{2, \dots, \mu\} \quad (3.51c)$$

$$\Pi_i^{[r_\zeta]} \Lambda_i \stackrel{(3.44)}{\leq} \frac{1-r}{1-r_1} \Lambda_i^{[r_1]} \quad (3.51d)$$

$$\Pi_i^{[r_k]} \stackrel{(3.45)}{\leq} \Pi_i^{[r_{k+1}]}, \quad \forall k \in \{\mu+1, \dots, \zeta-1\} \quad (3.51e)$$

$$\sum_{i=2}^{d_v} \Pi_i^{[r_k]} \Lambda_i \stackrel{(3.46)}{=} 1 - \frac{r}{r_k}, \quad \forall k \in \{\mu+1, \dots, \zeta\} \quad (3.51f)$$

$$0 \leq \Pi_i^{[r_k]} \leq 1, \quad \forall k \in \{\mu+1, \dots, \zeta\} \quad (3.51g)$$

$$\Lambda^{[r_k]}(1) = 1, \quad \Lambda_i^{[r_k]} \geq 0, \quad \forall k \in \{1, \dots, \mu\} \quad (3.51h)$$

$$R^{[r_k]}(x) \text{ from (3.15)}, \quad \forall k \in \{1, \dots, \mu\} \quad (3.51i)$$

$$\lambda_2^{[r_k]} \stackrel{(2.50)}{<} \frac{e^{r_k \tau(\Lambda^{[r_k]}, R^{[r_k]})}}{\sum_{j=2}^{d_c} \rho_j^{[r_k]}(j-1)}, \quad \forall k \in \{1, \dots, \mu\} \quad (3.51j)$$

$$\Pi_2^{[r_\zeta]} \stackrel{(3.38)}{<} \frac{1}{\lambda_2 \sum_{j=2}^{d_c} \rho_j(j-1)}. \quad (3.51k)$$

The constraints (3.51b)–(3.51f) are the five constraints ensuring rate-compatibility that were introduced above as (3.42)–(3.46). Furthermore, the trivial constraints (3.51g) and (3.51h) of all coefficients lying between 0 and 1 and the degree distributions' coefficients summing up to 1 are included. Constraint (3.51i) ensures all check node degree distributions to be in concentrated form. Finally, the stability condition (2.50) for the mother code rate and all shortened rates, as well as the novel puncturing stability condition (3.38) for the highest code rate r_ζ are included as constraints (3.51j) and (3.51k).¹¹

To enable reliable optimization results for arbitrary ranges of code rates, threshold computation is accomplished by DDE and the optimization problem is solved by the SA method presented in Section 2.7.2. Even though the search space of the multi-rate optimization problem is much larger than that of optimizing only a single shortened degree distribution, the heuristic SA approach still manages to find very good results. To reduce the optimization time, splitting the SA process into two phases as depicted in Figure 3.13 has proven useful:

- Left: Degree distributions $\Lambda^{[r_k]}$ ($k \in \{1, \dots, \mu\}$) and puncturing polynomials $\Pi^{[r_k]}$ ($k \in \{\mu+1, \dots, \zeta\}$) are jointly optimized at all rates subject to all constraints in (3.51). The current solution vector consisting of all polynomial coefficients is randomly modified in each step, and the cost function is evaluated. If an improvement is found, the modified vector will always be accepted as new current solution, while a degradation is only accepted with the probability given by (2.78).
- Right: Starting at the rate r_k that has the largest current gap to capacity γ_k

¹¹Due to constraint (3.51e), the puncturing stability condition is also automatically fulfilled for all punctured code rates if it is fulfilled for the highest code rate r_ζ .

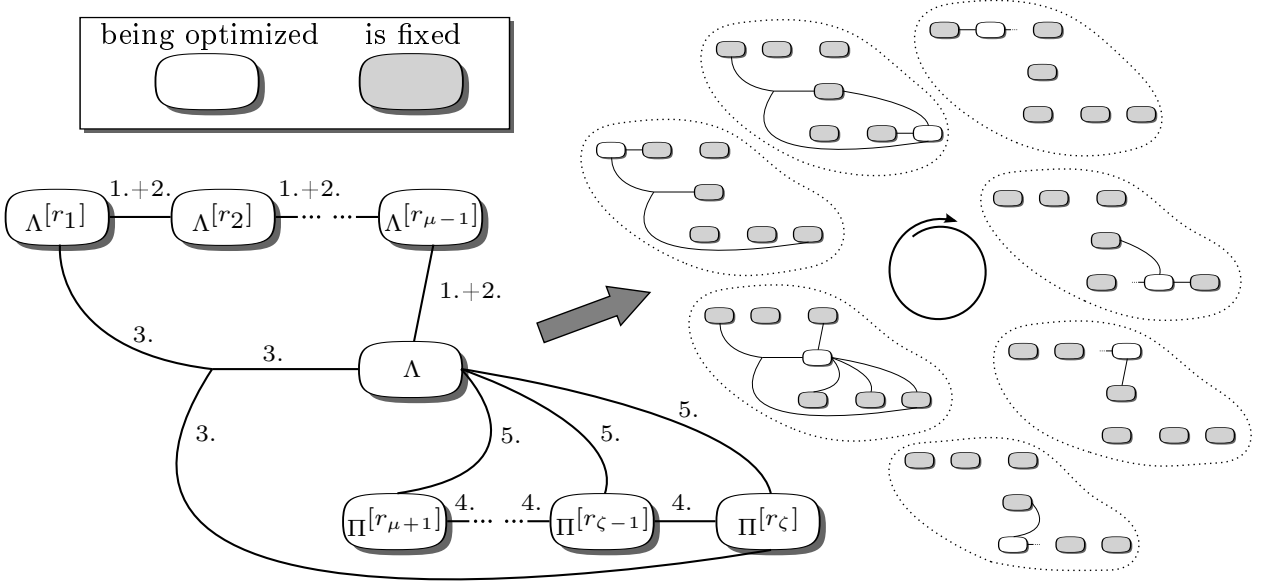


Figure 3.13: Active constraints of two-phase Simulated Annealing process.

and ending at the rate with the smallest current gap to capacity, the optimization process tries to improve each rate without violating any constraints in an iterative round-robin scheme.

The first phase tries to find the “deepest” local minimum by slowly cooling down the temperature T (i.e., the convergence parameter of SA) to avoid getting stuck in local minima. In the second phase, only downhill movements are allowed to make sure the final solution is maneuvered towards the bottom of the current local minimum.

Optimization Results

To confirm the effectiveness of the proposed optimization strategy, ensemble threshold results are presented for the challenging case of covering the range of (relevant) code rates from 0.1 to 0.9 in steps of 0.1 using only a single mother code and a maximum variable node degree of $d_v = 10$. It should be noted that these ensemble threshold results do not account for the reduction in effective relative code length since an infinitely large code length $N \rightarrow \infty$ is assumed for the threshold computation. For comparison, two different previous results from the literature are considered: the rate-compatible scheme presented in [TJ05] with the same set of code rates, the same maximum variable node degree, and a mother code rate of 0.5, as well as the results of [CRU01, Fig. 10] as a benchmark as to the best achievable performance for $d_v = 10$ if dedicated degree distributions are optimized for each of the target code rates without any rate-compatible features.

With the proposed scheme, multi-rate ensembles with mother codes of rates between 0.2 and 0.8 are optimized. Figure 3.14 shows the performance of five setups:

- M2: Proposed optimization with $r_\mu = 0.2$
- M5: Proposed optimization with $r_\mu = 0.5$

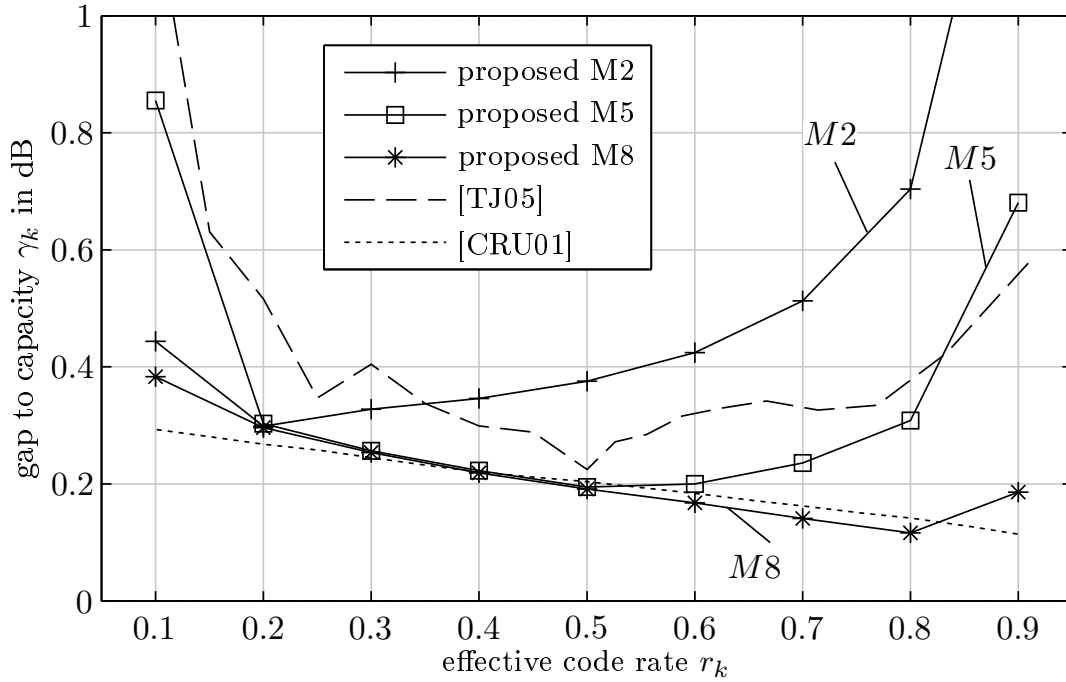


Figure 3.14: Gaps of ensemble thresholds to capacity over effective (shortened or punctured) code rate r_k .

- M8: Proposed optimization with $r_\mu = 0.8$
- [TJ05]: Reference results with $r_\mu = 0.5$
- [CRU01]: achievable gaps for independent codes without rate-compatibility.

Detailed optimization results of setups M5 and M8 are given in Table 3.5.

In Figure 3.14, the gaps $\gamma_k|_{\text{dB}}$ of the ensemble thresholds to capacity in dB as produced by the optimization algorithm are depicted for all effective code rates r_k . It can be seen that for the four rate-compatible setups, the respective minimal gap is achieved at the mother code rate and the gap increases when moving away from this rate. The values of the mean square gaps are given for all five curves. The proposed setup M5 outperforms the reference results [TJ05] with the same mother code rate in terms of the mean square gap, and also at each single effective code rate except 0.9. Setup M2 seems to strongly suffer from the fact that a large range of code rates from 0.3 to 0.9 has to be achieved by puncturing. The best performance of the rate-compatible schemes is achieved by setup M8 which is very close to the benchmark scheme [CRU01] which does not offer any rate-compatibility. As stated earlier, these results confirm that the proposed optimization strategy yields close to optimal performance since it comes very close to the case of unconstrained individual code optimization.

In Figure 3.15 the mean gaps $\bar{\gamma}_\mu|_{\text{dB}}$ computed by (3.47) as achieved by the proposed optimization algorithm for several mother code rates $r_\mu \in \{0.2, \dots, 0.8\}$ are shown. These results clearly suggest that the proposed optimization strategy favors mother code rates close to the upper limit of the supported effective code rate range, i.e., the resulting shortened codes seem to be more effective than the punctured ones. This

	degree	2	3	4	5	10	τ_k in dB	γ_k in dB
M5: $r_\mu = 0.5$	$\Lambda^{[0.1]}$	0.669	0.199	0.000	0.053	0.08	-0.43	0.86
	$\Lambda^{[0.2]}$	0.625	0.242	0.001	0.049	0.083	-0.66	0.30
	$\Lambda^{[0.3]}$	0.571	0.270	0.005	0.044	0.110	-0.36	0.26
	$\Lambda^{[0.4]}$	0.529	0.297	0.004	0.039	0.131	-0.02	0.22
	Λ	0.490	0.320	0.010	0.035	0.145	0.38	0.19
	$\Pi^{[0.6]}$	0.205	0.084	0.000	0.292	0.202	0.88	0.20
	$\Pi^{[0.7]}$	0.349	0.176	0.000	0.486	0.285	1.50	0.24
	$\Pi^{[0.8]}$	0.435	0.307	0.000	0.595	0.297	2.35	0.31
	$\Pi^{[0.9]}$	0.531	0.346	0.000	0.824	0.306	3.88	0.68
M8: $r_\mu = 0.8$	$\Lambda^{[0.1]}$	0.687	0.218	0.029	0.001	0.064	-0.90	0.38
	$\Lambda^{[0.2]}$	0.620	0.235	0.051	0.002	0.092	-0.67	0.30
	$\Lambda^{[0.3]}$	0.572	0.257	0.045	0.011	0.116	-0.36	0.25
	$\Lambda^{[0.4]}$	0.532	0.279	0.041	0.014	0.133	-0.02	0.22
	$\Lambda^{[0.5]}$	0.498	0.297	0.040	0.015	0.150	0.38	0.19
	$\Lambda^{[0.6]}$	0.460	0.328	0.033	0.012	0.167	0.85	0.17
	$\Lambda^{[0.7]}$	0.426	0.355	0.026	0.009	0.184	1.41	0.14
	Λ	0.390	0.386	0.017	0.006	0.200	2.16	0.12
	$\Pi^{[0.9]}$	0.136	0.098	0.343	0.034	0.071	3.38	0.19

Table 3.5: Optimized multi-rate ensembles for $r_\mu = 0.5$ and $r_\mu = 0.8$.

can be explained by the fact that in the considered asymptotic case (i.e., at infinite code length) the shortened codes do not suffer from any reduction in code length, and in terms of the ensemble thresholds do not behave differently from an un-shortened ensemble. In case of puncturing, however, there is an inherent loss in performance caused by the process of having to recover the punctured nodes. Consequently, in the asymptotic case, where only ensemble thresholds are considered, the optimal solutions favor multi-rate ensembles with many shortened codes over those with many

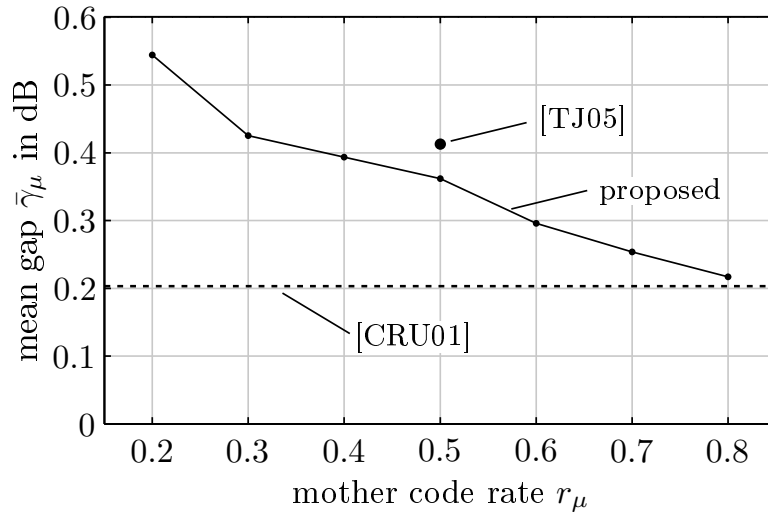


Figure 3.15: Mean gap to capacity $\bar{\gamma}_\mu$ achieved by the proposed optimization with different mother code rates r_μ .

punctured codes. The fully rate-compatible proposed scheme not only outperforms setup [TJ05] but even approaches the performance of the unconstrained optimization of individual codes according to [CRU01] (which is depicted as a flat line, as it does not have a dedicated mother code but consists of separately optimized codes) if a high rate mother code is used.

If concrete code realizations are considered instead of just the asymptotic ensemble thresholds, certainly, the optimal location of the mother code rate will be influenced by finite length effects such as, e.g., the reduced effective code length that was discussed above on pages 79 ff. These finite length effects are considered in the following section.

3.3.2 Optimized Code Construction

Given an optimized multi-rate ensemble described by its mother code degree distribution pair (Λ, R) , the shortened degree distribution pairs $(\Lambda^{[rs]}, R^{[rs]})$, and the puncturing polynomials $\Pi^{[rp]}(x)$ as described in the previous section, the problem of constructing an according multi-rate LDPC code consists in finding a check matrix that complies with all given parameters in a rate-compatible fashion (see Figures 3.11 and 3.12 on pages 81 and 83). This is achieved for the variable node degree distributions¹² by ensuring the check matrix to match the *rate-compatible variable node degree sequence* $\mathbf{d}_v^{\text{RC}} = (d_{v,1}^{\text{RC}}, \dots, d_{v,N}^{\text{RC}})$, which is defined analog to the mother degree sequence (2.85) but compiled according to Algorithm 5. An appropriate rounding needs to be incorporated, such that k_{start} and k_{end} are integer values and that the number of totally assigned degrees is correct for each code rate. The total number of entries of each degree is the same as in the *mother degree sequence* $\mathbf{d}_v$ given by (2.85), but the ordering of degrees is such that the correct shortened and punctured variable node degree distributions are achieved when bits are shortened starting at the lowest index and punctured starting at the highest index. The check matrix can either be first constructed by an arbitrary algorithm using the mother degree sequence and then have its columns reordered to match the rate-compatible degree sequence, or the construction algorithm can directly be constrained to yield a matrix with the desired column degree ordering. An exemplary mother degree sequence and the rate-compatible degree sequence according to Algorithm 5 are shown in Figure 3.16 for the multi-rate ensemble given in Table 3.5 with $r_\mu = 0.5$ and $N = 504$.

In the following, finite length effects influencing the performance of an LDPC code constructed with the PEG algorithm and a rate-compatible degree sequence will be discussed. Thereafter, a construction algorithm taking all discussed effects into account will be presented.

Guaranteed Recoverability

As described in Section 3.2.3, it is of utmost importance to ensure that all punctured nodes are recoverable. To achieve this, the rightmost $M_d = (1 - n_\zeta)N = \frac{r_\zeta - r_\mu}{r_\zeta}N$

¹²To also incorporate arbitrary (shortened) check node degree distributions, some more effort is required. Since here mostly concentrated check node degree distributions are considered, the following descriptions focus on this case only.

Algorithm 5 Compilation of rate-compatible degree sequence $\mathbf{d}_v^{\text{RC}}$.

```

1:  $n_k = \frac{1-r\mu}{1-r_k}$ , for  $k = 1, \dots, \mu$ 
2:  $n_k = \frac{r\mu}{r_k}$ , for  $k = \mu + 1, \dots, \zeta$ 
3:  $\Delta_{d'}^{[r_{k+1} \rightarrow r_k]} = \Lambda_{d'}^{[r_{k+1}]} \frac{1-r_k}{r_{k+1}-r_k} - \Lambda_{d'}^{[r_k]} \frac{1-r_{k+1}}{r_{k+1}-r_k}$ , for  $k = 1, \dots, \mu - 1$ ,  $d' = 2, \dots, d_v$ 
4: for  $k = \mu - 1, \dots, 1$  do
5:   for  $d = 2, \dots, d_v$  do
6:      $i_{\text{start}} = 1 + \sum_{k'=k+1}^{\mu-1} N(n_{k+1} - n_k) + \sum_{d'=2}^{d-1} \Delta_{d'}^{[r_{k+1} \rightarrow r_k]} N(n_{k+1} - n_k)$ 
7:      $i_{\text{end}} = \sum_{k'=k+1}^{\mu-1} N(n_{k'+1} - n_{k'}) + \sum_{d'=2}^d \Delta_{d'}^{[r_{k+1} \rightarrow r_k]} N(n_{k+1} - n_k)$ 
8:     for  $i = i_{\text{start}}, \dots, i_{\text{end}}$  do
9:        $d_{v,i}^{\text{RC}} = d$ 
10:    end for
11:  end for
12: end for
13: for  $d = 2, \dots, d_v$  do
14:    $i_{\text{start}} = (1 - n_1)N + 1 + \sum_{d'=2}^{d-1} \Lambda_{d'}^{[r_1]} n_1 N - \Pi_{d'}^{[r_\zeta]} \Lambda_{d'} N$ 
15:    $i_{\text{end}} = (1 - n_1)N + \sum_{d'=2}^d \Lambda_{d'}^{[r_1]} n_1 N - \Pi_{d'}^{[r_\zeta]} \Lambda_{d'} N$ 
16:   for  $i = i_{\text{start}}, \dots, i_{\text{end}}$  do
17:      $d_{v,i}^{\text{RC}} = d$ 
18:   end for
19: end for
20: for  $k = \zeta, \dots, \mu + 1$  do
21:   for  $d = 2, \dots, d_v$  do
22:      $i_{\text{start}} = n_\zeta N + 1 + \sum_{k'=k+1}^\zeta (n_{k'-1} - n_{k'}) N + \sum_{d'=2}^{d-1} (\Pi^{[r_k]} - \Pi^{[r_{k-1}]}) \Lambda_{d'} N$ 
23:      $i_{\text{end}} = n_\zeta N + \sum_{k'=k+1}^\zeta (n_{k'-1} - n_{k'}) N + \sum_{d'=2}^d (\Pi^{[r_k]} - \Pi^{[r_{k-1}]}) \Lambda_{d'} N$ 
24:     for  $i = i_{\text{start}}, \dots, i_{\text{end}}$  do
25:        $d_{v,i}^{\text{RC}} = d$ 
26:     end for
27:   end for
28: end for

```

positions in the rate-compatible degree sequence are marked as being punctured. For this purpose, let the boolean vector $\mathbf{p} = (\mathbf{p}_1, \dots, \mathbf{p}_N)$ contain a 0 at indices that are not punctured and a 1 at indices that are punctured:

$$\mathbf{p}_i = \begin{cases} 0, & \text{for } 1 \leq i < n_\zeta N \\ 1, & \text{for } n_\zeta N \leq i \leq N. \end{cases} \quad (3.52)$$

Recoverability can then be guaranteed for all punctured nodes by modifying the PEG algorithm to follow the rule introduced in Section 3.2.3 for guaranteed recoverability whenever edges are connected to a punctured variable node.

The originally proposed PEG algorithm uses a sorted degree sequence and thus connects variable nodes in a node-by-node manner in ascending variable node degree order. To obey this rule of ascending degrees, the following strategy can be employed.

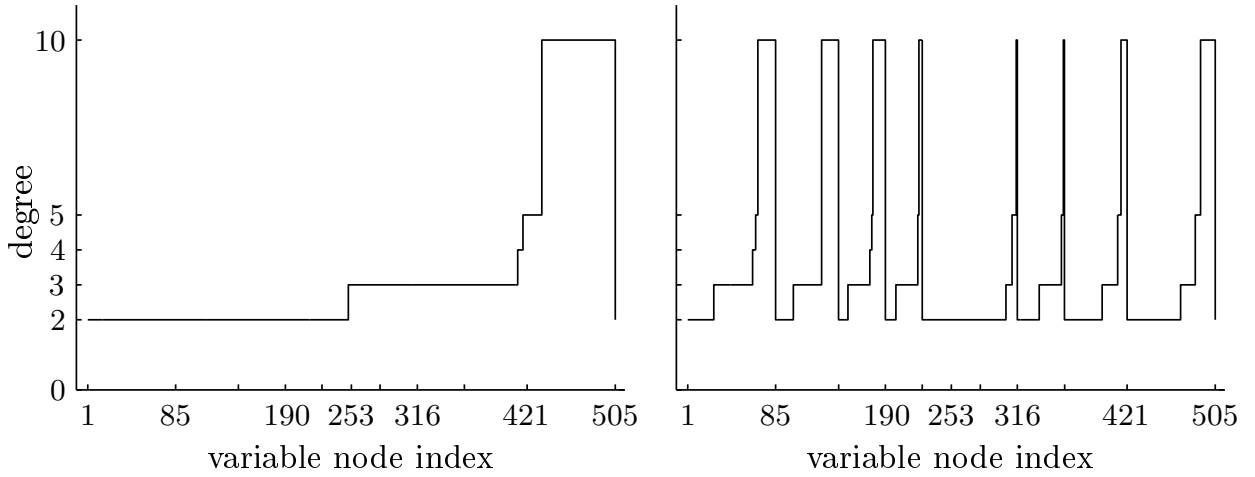


Figure 3.16: Mother degree sequence $\mathbf{d}_v$ (left) and rate-compatible degree sequence $\mathbf{d}_v^{\text{RC}}$ (right) of length $N = 504$.

First, let $\mathbf{P}_{\text{RC}}$ denote the $N \times N$ permutation matrix that converts the ascending mother degree sequence $\mathbf{d}_v$ into the rate-compatible degree sequence $\mathbf{d}_v^{\text{RC}}$:

$$\mathbf{d}_v^{\text{RC}} = \mathbf{d}_v \mathbf{P}_{\text{RC}}. \quad (3.53)$$

An intermediate check matrix $\tilde{\mathbf{H}}$ is now constructed by the modified PEG algorithm, using the mother degree sequence $\mathbf{d}_v$ (with ascending degrees) and the vector $\tilde{\mathbf{p}} = \mathbf{p} \mathbf{P}_{\text{RC}}^{-1}$, with $\mathbf{p}$ from (3.52), containing the permuted information about which nodes are punctured (since these are no longer simply the ones with highest indices). The final rate-compatible matrix $\mathbf{H}$ is then obtained by applying the permutation $\mathbf{P}_{\text{RC}}$ to the constructed matrix:

$$\mathbf{H} = \tilde{\mathbf{H}} \mathbf{P}_{\text{RC}}. \quad (3.54)$$

This way, the final check matrix has the desired rate-compatible degree sequence $\mathbf{d}_v^{\text{RC}}$ and the $(1 - n_\zeta)N$ nodes with highest indices are guaranteed to be recoverable.

Concentrated Shortened Check Node Degree Distributions

A major drawback of the method that was just presented is the check node degree distribution of the resulting shortened codes. While the desired variable node degree distribution is matched by the rate-compatible degree sequence, the check node degree distribution will have a form similar to that of a randomly shortened code given by (3.15) on page 55. Using this method, the shortened codes might even have degree-0 and degree-1 check nodes, which should be avoided in any case.¹³ This is due to the fact, that typically a large number of columns are swapped to transform the mother degree sequence into the rate-compatible degree sequence, resulting in a random-like selection of the shortened columns with respect to the matrix $\tilde{\mathbf{H}}$.

¹³These degree 0 and 1 check nodes can in most cases be easily avoided by performing a small number of additional column swaps of columns with equal degrees. However, none of the punctured columns can be swapped into the shortened parts, as they were explicitly constructed to ensure recoverability which could be violated by using other columns for the punctured positions.

Better performance is expected for concentrated check node degree distributions of the shortened codes, which was also assumed during the multi-rate optimization presented in Section 3.3.1. To achieve this, the following property of the PEG algorithm is exploited. Since the PEG algorithm works in a variable-node-by-variable-node manner and all check node degrees grow approximately uniformly, each submatrix $\mathbf{H}_{[1 \rightarrow N']}$ consisting of the N' columns of $\mathbf{H}$ that were processed first has a nearly concentrated check node degree distribution. Since here shortening is assumed to start at the lowest index, the shortened codes correspond to the submatrices $\mathbf{H}_{[(1-n_s)N+1 \rightarrow N]}$ of the columns with highest indices. These submatrices can be constructed to be check node concentrated by directly using the rate-compatible degree sequence $\mathbf{d}_v^{\text{RC}}$ and letting the PEG algorithm proceed in a node-by-node manner starting from the highest index and moving towards lower indices. This procedure is visualized in Figure 3.17.

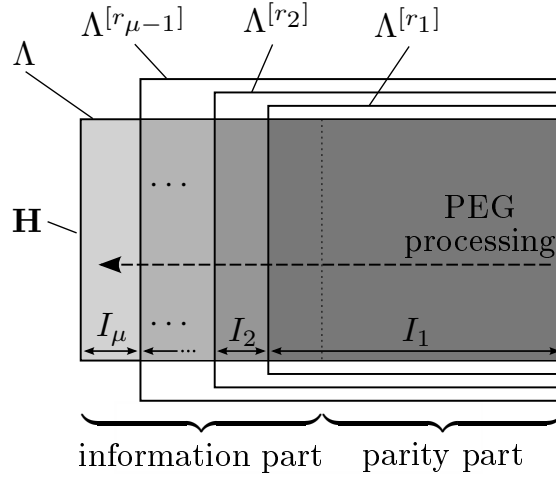


Figure 3.17: PEG processing starting from highest index to obtain check node concentrated shortened matrices. I_k denotes the range of indices added to extend the matrix of rate r_{k-1} to rate r_k (for $k \geq 2$).

Improving ACE Properties

The modified order of degrees within the rate-compatible degree sequence proposed above deviates from the ascending order of the originally proposed PEG algorithm. Empirical observations show that when creating a check matrix with the PEG algorithm, the order of degrees strongly influence the resulting ACE properties (cf. Section 2.3). To illustrate this behavior, five check matrices of dimension 504×1008 are constructed with the PEG algorithm using different degree orders:

- *ascending* degree order: as originally proposed,
- *descending* degree order: inverted order,
- *randomized* degree order: original degree sequence is randomly permuted,
- *rate-compatible* degree order: degree sequence from Algorithm 5,
- *iPEG + rate-compatible* degree order: degree sequence from Algorithm 5 and additionally the improved, ACE optimized PEG algorithm is used (cf. Section 2.8.2 on page 39).

The degree distribution is in all cases the one of rate 0.5 from the multi-rate ensemble given in Table 3.5 on page 88 with $r_\mu = 0.5$. Accordingly, the rate-compatible degree sequence was constructed using the distributions of this multi-rate ensemble. Exemplary degree sequences for all setups are depicted in Figure 3.18, where the last two setups use the same rate-compatible degree sequence. The ACE properties are described in terms of the ACE spectrum $\boldsymbol{\eta} = (\eta_4, \eta_6, \eta_8, \eta_{10})$ up to cycle length 10 (cf. Section 2.3). The ACE spectra of all setups are given in Figure 3.18 together with BER simulation results of all constructed check matrices after $I_{\max} = 100$ BP decoding iterations.

Since all codes have the same degree distribution, the BER curves are very similar in the waterfall region up to around 1.8 dB. In the error floor region, however, different behavior is observed for the different setups, which can be explained by the different ACE spectra. As all matrices have girth 6, there are no cycles of length 4 and, thus, by definition the first value in the ACE spectrum is ∞ . The randomized degree order shows the highest error floor, which agrees with its ACE spectrum having the smallest values (4, 0, 1), meaning that its cycles are most harmful as they have the lowest connectivity to nodes delivering extrinsic information. The rate-compatible ordering also has a poor ACE spectrum, but the absence of cycles with ACE 0 ($\eta_{rc,8} = 1 > 0 = \eta_{rand,8}$) already improves the error floor compared to the

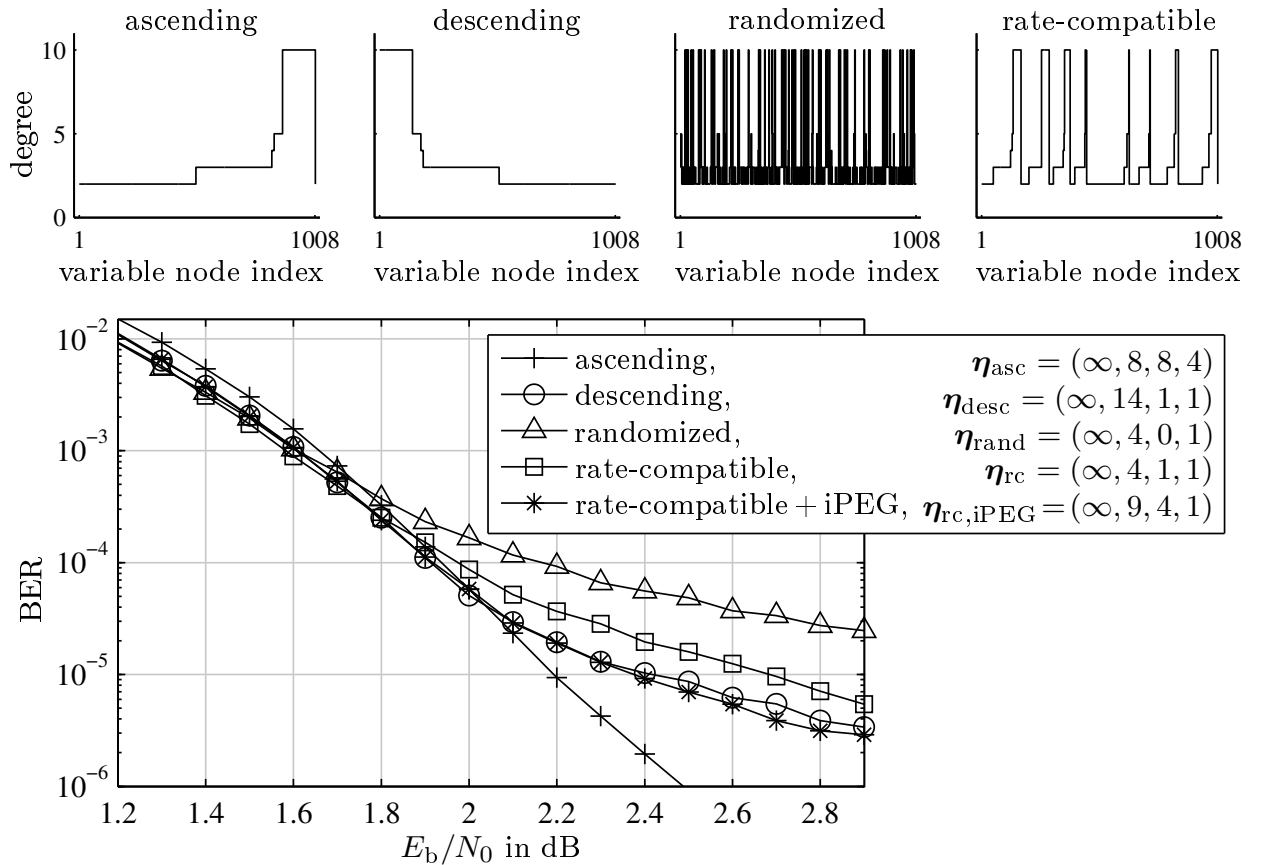


Figure 3.18: BER results after $I_{\max} = 100$ BP decoding iterations for PEG codes with $N = 1008$ and $r = 0.5$ for different degree sequence orderings. Additionally, the ACE spectra $\boldsymbol{\eta}$ of all check matrices are given.

randomized order. The descending order has the same minimum ACE value as the rate-compatible order for cycles of length 8 and 10, but the increased ACE value for cycles of length 6 ($\eta_{\text{desc},6} = 14 > 4 = \eta_{\text{rc},6}$) further improves the error floor as compared to the rate-compatible order. Another small gain is achieved by the rate-compatible order together with the iPEG rule which has a lower ACE value at cycle length 6 ($\eta_{\text{rc,iPEG},6} = 9 < 14 = \eta_{\text{desc},6}$), however, a higher value at cycle length 8 ($\eta_{\text{rc,iPEG},8} = 4 > 1 = \eta_{\text{desc},8}$). This agrees with the observation of [TJVW04] that longer cycles can be more detrimental than shorter cycles if their ACE values are smaller. Finally, the original ascending degree order exhibits the best error floor performance and has an ACE value of at least 4 for all considered cycle lengths.

Altogether, the rate-compatible degree order together with the iPEG improvement is the most suitable choice, since only the ascending order shows better error floor behavior but does not allow constructing rate-compatible codes with concentrated shortened degree distributions as described above.

Proposed Check Matrix Construction MuRiPEG

A novel check matrix construction algorithm termed MuRiPEG (*multi-rate improved PEG*) is introduced, taking into account all previously described finite length effects influencing the performance of the resulting multi-rate LDPC code. As input, it requires the desired mother code dimensions N and M as well as an optimized set of degree distributions and puncturing polynomials as produced by the above proposed asymptotic multi-rate optimization in Section 3.3.1. The resulting output is a mother code check matrix that can be shortened from the left and punctured from the right (cf. Figure 3.11 on page 81). A detailed description is given in Algorithm 6, which is explained in the following.

Firstly, Algorithm 5 is used to determine the rate-compatible degree sequence $\mathbf{d}_v^{\text{RC}}$ of length N from the degree distributions and puncturing polynomials of the optimized multi-rate ensemble. Since puncturing is to be done from the right, as many values at the end of the vector $\mathbf{p}$ are set to 1 as are needed to achieve the highest punctured code rate r_ζ .

The original goal of reordering the degrees was to achieve (nearly) check concentrated shortened codes. When shortened to the lowest rate r_1 , the effective check matrix consists of the rightmost $n_1 N = \frac{1-r_\mu}{1-r_1} N$ columns of the mother check matrix. For the ACE properties of this most shortened submatrix (index set I_1 in Figure 3.17 on page 92) it proves beneficial to reorder the degree sequence within these positions in ascending order, construct the matrix with ordered degrees, and at the end reorder the columns of the matrix according to the original rate-compatible degree sequence. This way, the same check concentration is achieved for all shortened submatrices, while simultaneously slightly improving the ACE properties due to a less random-like degree order. Thus, a permuted degree sequence $\hat{\mathbf{d}}_v^{\text{RC}} = \mathbf{d}_v^{\text{RC}} \mathbf{P}$ (lines 3–4) is used during PEG processing and afterwards the corresponding inverse permutation $\mathbf{P}^{-1}$ is applied to the constructed matrix (line 34). The vector $\mathbf{p}$ indicating which positions

Algorithm 6 Proposed MuRiPEG check matrix construction algorithm.

- 1: Input: $N, M, r_\mu = \frac{N-M}{N}, r_1 < \dots < r_\mu < \dots < r_\zeta, \mathbf{d}_v^{\text{RC}}$ from Algorithm 5
 - 2: Mark the last $\frac{r_\zeta - r_\mu}{r_\zeta} N$ positions as punctured in vector $\mathbf{p}$ (cf. (3.52))
 - 3: Compute $\tilde{\mathbf{d}}_v^{\text{RC}}$ by sorting the last $n_1 N = \frac{1-r_\mu}{1-r_1} N$ entries in $\mathbf{d}_v^{\text{RC}}$ in ascending order
 - 4: Let $\mathbf{P}$ denote the permutation matrix such that $\tilde{\mathbf{d}}_v^{\text{RC}} = \mathbf{d}_v^{\text{RC}} \mathbf{P}$
 - 5: Compute the permuted punctured positions $\tilde{\mathbf{p}} = \mathbf{p} \mathbf{P}$
 - 6: Initialize empty check matrix $\tilde{\mathbf{H}} = \mathbf{0}$
 - 7: Initialize check node degrees $\mathbf{d}_c = (d_{c,1}, \dots, d_{c,M}) = (0, \dots, 0)$
 - 8: Initialize counter for punctured nodes $c = 0$
 - 9: **for** $n = N, \dots, 1$ **do**
 - 10: **if** $\tilde{\mathbf{p}}_n = 1$ **then**
 - 11: $\tilde{H}_{n,M-c} = 1$ and $d_{c,M-c} = 1$
 - 12: $c = c + 1$
 - 13: **end if**
 - 14: **end for**
 - 15: Reset counter for punctured nodes $c = 0$
 - 16: **for** $n = N, \dots, 1$ **do**
 - 17: **if** $\tilde{\mathbf{p}}_n = 0$ **then**
 - 18: $\tilde{H}_{n,m} = 1$ and $d_{c,m} = d_{c,m} + 1$, with m randomly from $\arg \min_{\forall m'} d_{c,m'}$
 - 19: **end if**
 - 20: **for** $i = 2, \dots, \tilde{d}_{v,n}^{\text{RC}}$ **do**
 - 21: Initialize $\omega_m = \infty, \forall m$
 - 22: Expand $\mathcal{T}_n^l$ up to depth l , setting $\omega_m = j$ for all check nodes m that are reached for the first time at depth j , until no more nodes can be added
 - 23: During tree expansion, also compute ACE_m for all reachable check nodes m as the minimum ACE value of any shortest cycle passing through variable node n and check node m assuming an edge was placed between n and m
 - 24: **if** $\tilde{\mathbf{p}}_n = 1$ **then**
 - 25: Set $\omega_m = 0, \forall m > M - c$ to ensure recoverability
 - 26: **end if**
 - 27: $\mathcal{K}_{\text{girth}} = \arg \max_{\forall m'} \omega_{m'}$
 - 28: $\mathcal{K}_{\text{weight}} = \arg \min_{m' \in \mathcal{K}_{\text{girth}}} d_{c,m'}$
 - 29: $\mathcal{K}_{\text{ACE}} = \arg \max_{m' \in \mathcal{K}_{\text{weight}}} \text{ACE}_{m'}$
 - 30: $\tilde{H}_{n,m} = 1$ and $d_{c,m} = d_{c,m} + 1$, with m randomly from $\mathcal{K}_{\text{ACE}}$
 - 31: **end for**
 - 32: **if** $\tilde{\mathbf{p}}_n = 1$ **then** $c = c + 1$
 - 33: **end for**
 - 34: The output is then given by the permuted matrix $\mathbf{H} = \tilde{\mathbf{H}} \mathbf{P}^{-1}$
-

correspond to punctured nodes also needs to be permuted by $\tilde{\mathbf{p}} = \mathbf{p}\mathbf{P}$ to ensure that the punctured nodes are in the rightmost positions of the final check matrix.

Before the actual PEG processing (lines 16–33) starts, the following pre-initialization (lines 9–14) is done to simplify the construction of guaranteed recoverable punctured nodes. Starting at the highest variable node index and moving towards lower indices, whenever a punctured column is reached, a 1 is placed in the row with highest index that does not yet have a 1 entry (i.e., the unique guaranteed survived check node that is connected to each punctured node). Later, during PEG processing of punctured columns, no 1 is allowed to be placed at higher indices than this pre-initialized position, assuring guaranteed recoverability. In Algorithm 6 this is realized by setting the “distance” ω_m to all check nodes m with indices higher than the one of the pre-initialized row to zero, avoiding any further connection to them (line 25).

The actual PEG processing starts at line 16. For unpunctured columns, the first 1 entry is placed in a random row m of minimum current weight $d_{c,m}$. This is not required for punctured columns, as their first 1 entry was already placed in the pre-initialization step. After that, the remaining $\tilde{d}_{v,n}^{\text{RC}} - 1$ entries are placed one-by-one according to the following criteria. From the set $\mathcal{K}_{\text{girth}}$ of all check nodes m' that have maximum distance $\omega_{m'}$ to the current variable node (or are not reachable, i.e., $\omega_{m'} = \infty$), the ones with minimum current degree $d_{c,m'}$ are selected as candidates $\mathcal{K}_{\text{weight}}$. Then, from all candidates in $\mathcal{K}_{\text{weight}}$ that additionally have the maximum ACE value $\eta_{m'}$ (set $\mathcal{K}_{\text{ACE}}$) one is chosen randomly for connection.

3.3.3 Simulation Results

For the evaluation of the proposed multi-rate ensemble optimization (Section 3.3.1) together with the novel matrix construction algorithm (Section 3.3.2), extensive BER simulations are conducted. The parameters of the multi-rate setup are chosen as:

- supported code rates $r_k \in \{r_1, r_2, \dots, r_9\} = \{0.1, 0.2, \dots, 0.9\}$,
- mother code rate $r_\mu \in \{0.2, 0.3, \dots, 0.8\}$, i.e., $\mu \in \{2, 3, \dots, 8\}$,
- maximum variable node degree $d_v = 10$,
- concentrated check node degree distributions,
- mother code length $N = 1008$.

In the beginning of Section 3.3 it was shown that the optimal mother code position in terms of maximum average relative code length $\bar{n}_{\text{eff}}$ lies close to the center of the target code rate range (cf. (3.40) and (3.41) and Figure 3.10). In contrast, a result of the asymptotic multi-rate optimization in Section 3.3.1 (i.e., for code length $N \rightarrow \infty$) was that the mother code rate should be chosen close to the upper limit of the code rate range to minimize the asymptotic performance degradation of recovering punctured nodes. Consequently, at finite code lengths, both of these effects can be expected to superpose, such that the optimal mother code position in terms of BER performance lies somewhere in between and needs to be determined empirically at each mother

code length N . To this end, seven multi-rate ensembles with seven different mother code rates $r_\mu \in \{0.2, 0.3, \dots, 0.8\}$ and otherwise identical parameters are optimized as described in Section 3.3.1. The mean gaps $\bar{\gamma}_\mu$ to capacity of the ensemble thresholds achieved by these seven multi-rate ensembles are shown in Figure 3.15 on page 88.

For the construction of check matrices, three different setups are considered:

ascPEG : the original PEG algorithm with ascending degree order is used to construct the mother check matrix of the optimized multi-rate ensemble. To obtain useful punctured codes for very high code rates, the guaranteed recoverability from Section 3.2.3 is incorporated into the PEG processing. After construction, the check matrix columns are reordered to match the rate-compatible degree sequence $\mathbf{d}_v^{\text{RC}}$.

MuRiPEG : the proposed MuRiPEG algorithm (Algorithm 6) is used to construct the mother check matrix of the optimized multi-rate ensemble.

dedicated : as reference, this setup uses one dedicated, optimized code for each code rate without any rate adaptation. For this purpose, for each code rate one degree distribution with the same maximum variable node degree as for the multi-rate ensemble is optimized. Using these degree distributions and the original PEG algorithm, one dedicated check matrix of code rate r_k and length $n_k N$ is constructed, where N and r_μ are the length and rate of the mother code in the multi-rate setup and $n_k N$ corresponds to the effective (shortened or punctured) code length in the multi-rate setup.

Each of these three setups is employed at each mother code rate rate r_μ . Then, one BER simulation with $I_{\text{max}} = 100$ BP decoding iterations is conducted at each target code rate r_k (using shortening or puncturing in the first two setups and the respective dedicated code in the third setup). This results in 189 BER curves, one for each combination of matrix construction method, mother code rate, and target code rate. For easier comparison of the different approaches, for each BER curve the BER threshold $\tau_{\mu,k}^{[10^{-4}]}$ is determined, denoting the E_b/N_0 value at which BER 10^{-4} is reached¹⁴ for mother code rate r_μ and effective code rate r_k . The resulting gap of the BER threshold to the respective capacity limit in dB is denoted (cf. Section 2.6.2)

$$\gamma_{\mu,k}^{[10^{-4}]} \Big|_{\text{dB}} = \tau_{\mu,k}^{[10^{-4}]} \Big|_{\text{dB}} - \mathbf{q}(r_k) \Big|_{\text{dB}}. \quad (3.55)$$

Furthermore, to compare the different multi-rate ensembles, for each mother code rate and each matrix construction method, the mean gap of the BER thresholds to capacity in dB is computed by

$$\bar{\gamma}_\mu^{[10^{-4}]} \Big|_{\text{dB}} = \frac{1}{9} \sum_{k=1}^9 \gamma_{\mu,k}^{[10^{-4}]} \Big|_{\text{dB}}. \quad (3.56)$$

¹⁴The BER of 10^{-4} is chosen to primarily compare the waterfall performance. Some of the MuRiPEG codes exhibit error floors starting around BER 10^{-4} and some ascPEG codes even show error floors at considerably higher BERs.

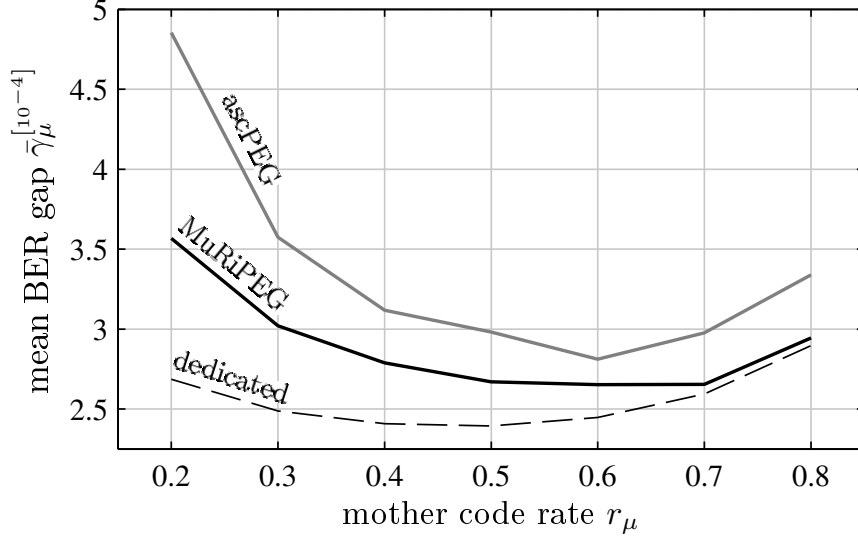


Figure 3.19: Mean gaps $\bar{\gamma}_\mu^{[10^{-4}]}$ to capacity at BER 10^{-4} after $I_{\max} = 100$ BP decoding iterations for different mother code rates r_μ .

Using this measure, the performance of the three matrix construction methods is compared in Figure 3.19 for all mother code rates r_μ . The novel MuRiPEG approach clearly outperforms the ascPEG approach at all mother code rates with improvements between 0.16 dB and 1.29 dB in terms of mean gap to capacity. While the ascPEG curve exhibits a distinct minimum of $\bar{\gamma}_6^{[10^{-4}]}|_{\text{dB}} = 2.81$ dB at mother code rate $r_\mu = 0.6$, the MuRiPEG curve is less bent and achieves almost constant minimal mean gaps of about $\bar{\gamma}_\mu^{[10^{-4}]}|_{\text{dB}} = 2.65$ dB at mother code rates $r_\mu \in \{0.5, 0.6, 0.7\}$. Furthermore, at mother code rates 0.7 and 0.8 the MuRiPEG approach on average performs less than 0.06 dB away from the reference setup that uses dedicated codes at each rate. As expected, the reference setup achieves its minimum value of $\bar{\gamma}_5^{[10^{-4}]}|_{\text{dB}} = 2.39$ dB at mother code rate $r_\mu = 0.5$, as it does not suffer from any puncturing penalty and thus benefits from the maximum average code length $\bar{n}_{\text{eff}}$ at the center of the code rate range.

A more detailed comparison of all setups is provided in Figure 3.20, including all BER over E_s/N_0 curves (larger plots) and the respective BER gaps to capacity $\gamma_{\mu,k}^{[10^{-4}]}|_{\text{dB}}$ over the effective code rate (smaller plots). Here, it is especially noticeable that very early error floor behavior manifests in many cases for the ascPEG approach. Moreover, it can be seen that the MuRiPEG outperforms the ascPEG especially at low code rates, as it benefits from the concentrated check node degree distributions of the shortened codes.

3.4 Concluding Remarks

In this chapter, a highly flexible design methodology for multi-rate LDPC codes based on a single mother code with information shortening and parity puncturing was explored. It includes the asymptotic threshold optimization on degree distribution level

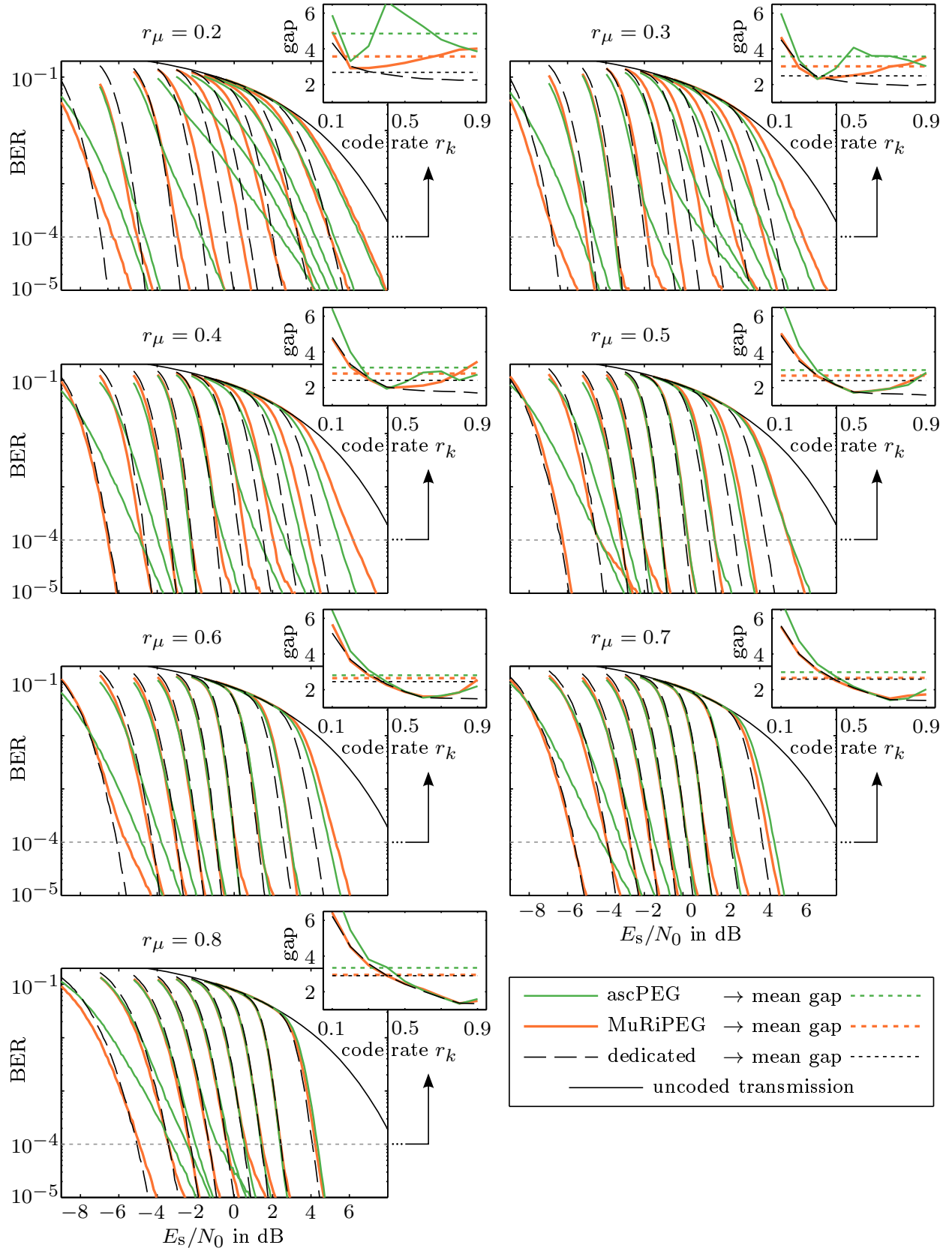


Figure 3.20: BER over E_s/N_0 curves for the setups ascPEG, MuRiPEG, and dedicated for all mother code rates r_μ and all target code rates r_k . The smaller plots show the gaps $\gamma_{\mu,k}^{[10^{-4}]}|_{\text{dB}}$ of the BER curves to capacity in dB at BER 10^{-4} over the effective code rate r_k .

and also the construction of a concrete parity check matrix. While the asymptotic results suggest that close-to-capacity error correction performance can be achieved by choosing the mother code rate close to the upper limit of the code rate range, finite length considerations show that the optimal position depends on the chosen range of target code rates and also the mother code length N . For the challenging case of covering the wide code rate range from 0.1 to 0.9 with a relatively small mother code length of $N = 1008$ bits, it was shown that with only a single mother code the proposed approach can perform within 0.06 dB of the case of using one dedicated code for each code rate. For smaller ranges of code rates (e.g., 1/2 to 5/6 as used in WLAN and WiMAX) the proposed method can be expected to perform even better, as performance degradation is especially observed at code rates that are far away from the mother code rate. Due to limited space, however, no further simulation results are shown here.

3.4.1 Comparison to other Multi-Rate Schemes

In this section, the performance of the proposed MuRiPEG scheme is compared to several results from the literature. Only cases of multi-rate LDPC codes that are based on a single mother code are considered. The gaps $\gamma_{\text{BER}}^{[10^{-4}]}$ to the capacity limit of the error rate curves at BER 10^{-4} that are achieved by different methods at different effective code rates are compared (solid lines in Figure 3.21). In cases where only FER curves are available, gaps $\gamma_{\text{FER}}^{[10^{-2}]}$ to the capacity limit are determined at FER 10^{-2} (dashed lines in Figure 3.21). Since for different methods the effective code rate has a different impact on the effective code length (shortening and puncturing both decrease the effective code length, whereas *extension*¹⁵ increases the effective code length), it is not possible to compare all results with exactly the same code lengths. However, the results are split into three plots, each of which only contains setups with approximately the same average code lengths. The top plot in Figure 3.21 contains setups with mother code length $N \approx 1024$, the center plot shows setups with average code lengths $\bar{N} \approx 1920$ (for MuRiPEG the mother code length is chosen such that the same average code length is obtained within the range of code rates supported by the reference schemes), and the bottom one compares two setups with mother code length $N = 10080$. The considered schemes are summarized in Table 3.6.

The proposed *MuRiPEG* scheme outperforms all of the other schemes for nearly all configurations. It is outperformed only at some single effective code rates, which is mostly caused by a higher effective code length of the reference scheme at that particular code rate. At the same time, it is the most flexible scheme allowing for an arbitrary choice of code rate range, mother code rate and code length. The only other scheme that covers the whole relevant code rate range is [TJ05], which is, however, considerably inferior to the proposed scheme especially at high code rates. Furthermore, the

¹⁵A mother code's check matrix can be *extended* by an appropriate number M_e of rows and columns to decrease the code rate from $r_\mu = \frac{N-M}{N}$ to $r_e = \frac{N-M}{N+M_e}$. At the same time the effective code length is increased from N to $N_e = N + M_e$.

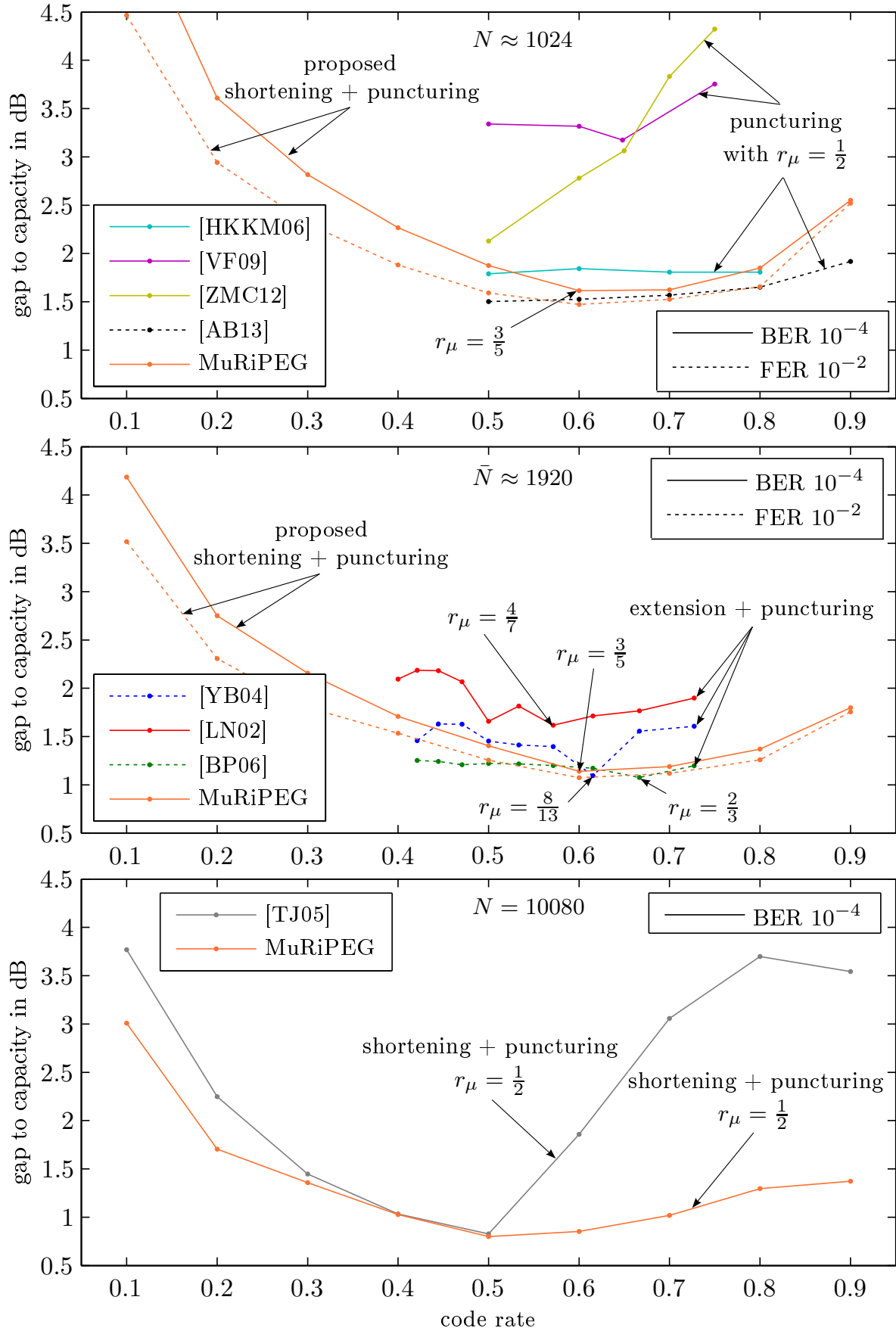


Figure 3.21: Gap to capacity of BER/FER curves for the proposed method and several multi-rate schemes from the literature (cf. Table 3.6).

	r_μ	N	code rate range	lower code rates	higher code rates
[HKKM06]	$\frac{1}{2}$	1024	$\frac{1}{2} \dots \frac{4}{5}$	-	puncturing
[VF09]	$\frac{1}{2}$	1000	$\frac{1}{2} \dots \frac{3}{4}$	-	puncturing
[ZMC12]	$\frac{1}{2}$	1008	$\frac{1}{2} \dots \frac{3}{4}$	-	puncturing
[AB13]	$\frac{1}{2}$	1024	$\frac{1}{2} \dots \frac{9}{10}$	-	puncturing
MuRiPEG	$\frac{3}{5}$	1008	$\frac{1}{10} \dots \frac{9}{10}$	shortening	puncturing
[LN02]	$\frac{4}{7}$	1792	$\frac{2}{5} \dots \frac{8}{11}$	extension	puncturing
[YB04]	$\frac{8}{13}$	1664	$\frac{8}{19} \dots \frac{4}{5}$	extension	puncturing
[BP06]	$\frac{2}{3}$	1536	$\frac{8}{19} \dots \frac{4}{5}$	extension	puncturing
MuRiPEG	$\frac{3}{5}$	2310	$\frac{1}{10} \dots \frac{9}{10}$	shortening	puncturing
[TJ05]	$\frac{1}{2}$	10080	$\frac{1}{10} \dots \frac{9}{10}$	shortening	puncturing
MuRiPEG	$\frac{1}{2}$	10080	$\frac{1}{10} \dots \frac{9}{10}$	shortening	puncturing

Table 3.6: Several multi-rate schemes based on single LDPC mother codes.

proposed scheme can even easily be extended to yield QC LDPC codes, which makes it also suitable for the design of codes for practical applications.¹⁶

3.4.2 Special Consideration of Non-Binary LDPC Codes

In general, all methods presented in this chapter are also applicable to non-binary LDPC codes if a suitable density evolution implementation for the threshold computation of non-binary codes is used. However, it was shown by multiple authors that in the non-binary case shortening (puncturing) can be improved by distributing single shortened (punctured) bits over the binary representation of multiple Galois field symbols, instead of shortening (puncturing) the symbols as a whole [KHM08, ZBX10, GSD10]. More recently, in [KDPS10, KDPS11, PDN14] *multiplicative repetition* was introduced. This technique starts with a single high rate mother code and achieves arbitrarily low code rates by simply repeating code words and multiplying them with pseudo-random Galois field elements. Surprisingly, this very simple scheme yields low-rate codes that are better than any currently known low-rate binary LDPC codes [KDPS11].

¹⁶This extension to the case of QC LDPC code is included in the multi-length optimization presented in Section 4.2.4.

Multi-Length LDPC Codes

Besides operating at multiple code rates, for most communication systems it is also advantageous to operate at multiple code lengths. In principle, long codes are desirable as they can achieve better error correction performance than shorter codes. At the same time, however, the length of a block code inherently implicates a certain delay caused by the block-wise processing of code words. For applications with delay constraints – and especially for those with low transmission data rates – the code length can be limited to a certain maximum value. Figure 4.1 exemplarily shows the inherent algorithmic delay ($= \frac{\text{code length} \cdot \text{code rate}}{\text{data rate}}$) that results from using a block code of length N for different code rates r and data rates. A typical data rate of $12.2 \frac{\text{kBit}}{\text{s}}$ for voice communication is given by the highest mode of the *Adaptive Multi-Rate* (AMR) speech codec [Eur00] that is used in the *Global System for Mobile Communications* (GSM) and *Universal Mobile Telecommunications System* (UMTS) cellular mobile networks. A maximum tolerable delay of 150 ms is often assumed for voice communication, resulting in a maximum code length around a couple of thousand bits for this application, depending on the employed code rate.

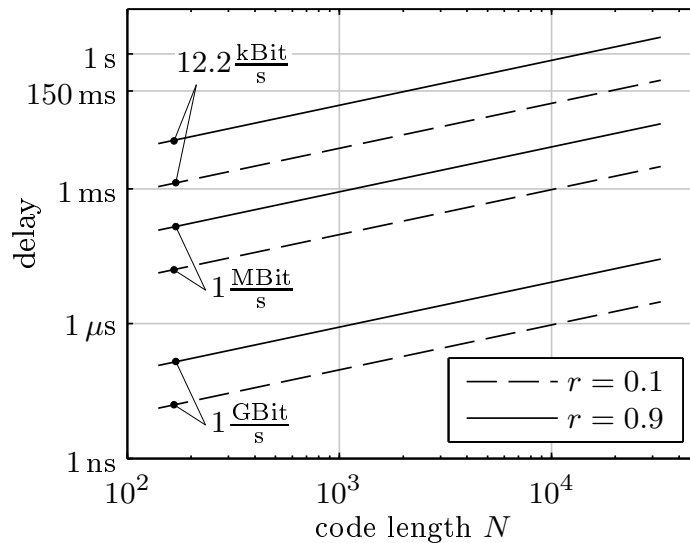


Figure 4.1: Algorithmic delay of block channel codes for different data rates, code rates, and code lengths.

The actual end-to-end delay of a transmission system can be even higher due to additional interleaving, the employed source coding, the signal propagation time itself, and the time required for signal processing, resulting in an even smaller applicable code length. Additional reasons for using short codes can be limited memory availability and low computational power, that prohibit storing and processing long code words. To account for low delay and low data rate applications (or cheap, low complexity transceivers) and still enable increased error correction performance for delay insensitive and/or high data rate applications (or more advanced transceivers) it is essential for the forward error correction functionality of a communication system to offer a range of possible code lengths.

Traditionally, the research on LDPC codes is divided into the *asymptotic* analysis of (very long) codes on the one hand, and the consideration of *finite length* effects that are predominant at short code lengths on the other hand. However, since LDPC codes are block codes, they have an inherently fixed code length and most of the researched code design methods are specific to one certain code length. To still support multiple code lengths, current systems either specify separate LDPC codes (e.g., WLAN and DVB) or use quasi-cyclic (QC) LDPC codes that are expanded by different lifting factors (e.g., WiMAX). An exemplary overview of the ranges of supported code lengths and the total numbers of specified LDPC codes of different systems is given in Table 4.1.

As in the multi-rate case of the previous chapter, here, a multi-length code is considered as an LDPC mother code that supports multiple code lengths and can be encoded and decoded with a single encoder and decoder implementation. To construct such multi-length LDPC codes, two approaches are considered in the following. First, *concurrent information shortening and parity puncturing* is investigated as a method to reduce the code length of an arbitrary LDPC mother code of fixed code rate. The second approach is based on a joint optimization of a QC LDPC mother code of fixed dimension for multiple lifting factors to achieve different code lengths of the expanded check matrices.

	number of LDPC codes	supported code lengths	uncoded lengths
IEEE 802.16e (WiMAX) [IEE09a]	6	576 – 2304	288 – 1920
IEEE 802.11n (WLAN) [IEE09b]	12	648 – 1944	324 – 1620
DVB-T2 [Eur09b]	13	16200, 64800	3240–54000
DVB-S2 [Eur09a]	22	16200, 64800	3240–58320

Table 4.1: Numbers and code lengths of LDPC codes specified in different communication standards.

4.1 Length Reduction by Concurrent Shortening and Puncturing

Information shortening and parity puncturing were introduced in the previous chapter as techniques to flexibly adapt the code rate of a fixed rate mother code. In contrast to adapting the code rate, in this section *concurrent shortening and puncturing* is considered to reduce the effective code length while keeping the code rate constant. A similar approach was proposed in [KCY12] to adapt the length of irregular repeat-accumulate (IRA) codes. While very good performance is achieved for this class of codes, the method of [KCY12] is not generally applicable to other kinds of LDPC codes and is, thus, not further considered in this section.¹

In the following the general process of concurrent shortening and puncturing is introduced by explaining the encoding and decoding steps. After this, an asymptotic analysis of the achievable ensemble threshold performance is provided and a *length-compatible* joint optimization problem for finding shortened degree distributions and puncturing polynomials for length reduction is proposed. The effectiveness of the proposed method is confirmed by comparing ensemble threshold as well as BER performance simulation results to that of dedicated codes that are specifically optimized for each supported code.

4.1.1 Encoding and Decoding

When a mother code of rate $r = \frac{K}{K+M}$ is shortened to the lower code rate $r_s = \frac{K_s}{K_s+M}$, at the same time according to (3.2) the effective number of information bits is reduced to

$$K_s \stackrel{(3.2)}{=} k_s K = \frac{r^{-1} - 1}{r_s^{-1} - 1} K.$$

Likewise, when the mother code is punctured to the higher code rate $r_p = \frac{K}{K+M_p}$, according to (3.21) the number of transmitted parity bits is reduced to

$$M_p \stackrel{(3.21)}{=} m_p M = \frac{r_p^{-1} - 1}{r^{-1} - 1} M. \quad (4.1)$$

By combining these two effects, the resulting relative code word length is reduced to

$$n_\ell := \frac{N_\ell}{N} = \frac{k_s K + m_p M}{N}. \quad (4.2)$$

To ensure the resulting code rate

$$r' = \frac{k_s K}{k_s K + m_p M} \quad (4.3)$$

¹As the method of [KCY12] is restricted to a special class of codes it will instead be compared to the approach presented in Section 4.2 which is also only applicable to a special class of codes.

after both shortening and puncturing to be equal to the mother code rate $r = \frac{K}{K+M}$, the following constraint must be fulfilled:

$$\begin{aligned} r' &\stackrel{!}{=} r &\Leftrightarrow & \frac{k_s K}{k_s K + m_p M} = \frac{K}{K + M} \\ & &\Leftrightarrow & k_s K^2 + k_s K M = k_s K^2 + m_p K M \\ & &\Leftrightarrow & k_s = m_p. \end{aligned} \quad (4.4)$$

Together with (4.2), this results in the rule that reducing the relative code length to n_ℓ is achieved by concurrently shortening and puncturing the same fractions $k_s = m_p = n_\ell$ of the information and parity words, respectively. The *virtual* shortened and punctured code rates that would result if only either one of the two operations were conducted alone are given by

$$r_{s,\ell} = \frac{n_\ell K}{n_\ell K + M} = \frac{n_\ell}{n_\ell + r^{-1} - 1} \quad (4.5)$$

for shortening and

$$r_{p,\ell} = \frac{K}{K + n_\ell M} = \frac{r}{r + n_\ell(1 - r)} \quad (4.6)$$

for puncturing, respectively. Thus, each reduced relative code length n_ℓ that should be achieved, can be represented by a pair $(r_{s,\ell}, r_{p,\ell})$ of shortened and punctured code rates which need to be realized concurrently. Using these virtual code rates, encoding and decoding of a length-reduced code word exactly follow the descriptions of shortening and puncturing given in Section 3.1.1 and Section 3.2.1, respectively. Shortened positions are initialized with zeros before encoding and the corresponding systematic positions as well as the punctured positions are discarded before transmission. In the decoder, shortened positions are initialized with LLRs $+\infty$ and punctured positions with LLRs 0. This way, for both encoding and decoding the mother code implementation can be used without modifications.

The exemplary code word structures of a rate-2/3 mother code and code words with lengths reduced to $n_3 = 0.75$, $n_2 = 0.5$, and $n_1 = 0.25$ of the original length are shown in Figure 4.2. For each length, also the respective code rate pair $(r_{s,\ell}, r_{p,\ell})$ is given. In analogy to the rate-compatible scheme from Section 3.3 (cf. Figure 3.11 on page 81), the reduction of the code word length is done in a *length-compatible* fashion, meaning that bits are shortened from left and punctured from right successively. As in the rate-compatible case of the previous chapter, this is either enabled by reordering the check matrix columns according to the positions that should be shortened and punctured or by directly constructing the check matrix of the mother code to have all potentially shortened positions on the left and all potentially punctured positions on the right.

4.1.2 Asymptotic Analysis and Optimization

For concurrent utilization of shortening and puncturing, the individual asymptotic analysis of message-passing decoding for both techniques from Sections 3.1.2 and 3.2.2

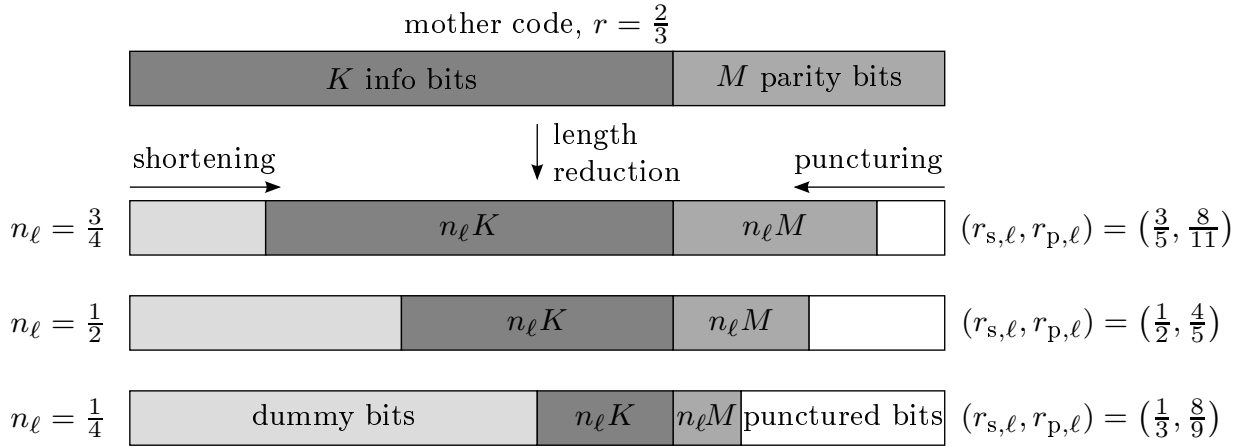


Figure 4.2: Exemplary code word structure of mother code with rate $2/3$ and length reduced to $n_3 = 0.75$, $n_2 = 0.5$, and $n_1 = 0.25$.

is still valid. The shortening of variable nodes (and connected edges) can still be modeled by simply removing them from the Tanner graph, since the additional presence of erased messages (LLRs 0 at the punctured variable nodes) does not change the effect of the perfect knowledge (LLRs $+\infty$ at shortened variable nodes). Consequently, the asymptotic behavior of the proposed length reduction can be described by first applying shortening to the mother code and then puncturing the resulting intermediate shortened code. The ensemble threshold τ of the length-reduced code can thus be computed using the density evolution for punctured codes as introduced in Section 3.2.2 on pages 68 ff.

Threshold Computation

In accordance with Chapter 3, the variable node degree distribution (in node perspective) of the mother code is denoted $\Lambda(x)$, the shortened degree distribution at virtual code rate $r_{s,\ell}$ is denoted $\Lambda_i^{[r_{s,\ell}]}(x)$, and the puncturing polynomial of virtual code rate $r_{p,\ell}$ is denoted $\Pi_i^{[r_{p,\ell}]}(x)$. For the computation of the ensemble threshold after length reduction, the puncturing polynomial first needs to be rescaled as follows. According to its definition, the coefficient $\Pi_i^{[r_{p,\ell}]}$ denotes the fraction of degree i nodes that are punctured in the mother code, resulting in a total number of $\Pi_i^{[r_{p,\ell}]} \Lambda_i N$ punctured degree i nodes. The shortened code which is to be punctured here, however, has a different total number of nodes $N_s < N$ and also potentially different fractions $\Lambda_i^{[r_{s,\ell}]}$ of nodes of each degree i . To still puncture the correct total number of nodes, the coefficients of $\Pi_i^{[r_{p,\ell}]}(x)$ are rescaled according to

$$\begin{aligned}
 \check{\Pi}_i^{[n_\ell]} \Lambda_i^{[r_{s,\ell}]} N_s &\stackrel{!}{=} \Pi_i^{[r_{p,\ell}]} \Lambda_i N \\
 \Leftrightarrow \quad \check{\Pi}_i^{[n_\ell]} &= \Pi_i^{[r_{p,\ell}]} \frac{\Lambda_i}{\Lambda_i^{[r_{s,\ell}]}} \frac{N}{N_s} = \frac{\Pi_i^{[r_{p,\ell}]} \Lambda_i}{\Lambda_i^{[r_{s,\ell}]} (1 - (1 - n_\ell) r)}, \quad (4.7)
 \end{aligned}$$

resulting in the scaled puncturing polynomial

$$\check{\Pi}^{[n_\ell]}(x) = \prod_{i=2}^{d_v} \check{\Pi}_i^{[n_\ell]} x^i. \quad (4.8)$$

The ensemble threshold $\tau(\Lambda^{[r_{s,\ell}]}, R^{[r_{s,\ell}]}, \check{\Pi}^{[n_\ell]})$ of the length-reduced code can then be computed by density evolution of the punctured ensemble $(\Lambda^{[r_{s,\ell}]}, R^{[r_{s,\ell}]}, \check{\Pi}^{[n_\ell]})$ using the Gaussian approximation (GA) or the discretized density evolution (DDE) from Section 3.2.2. The check node degree distribution $R^{[r_{s,\ell}]}(x)$ depends on the exact selection of columns that are shortened and will take the binomial form (3.15) if a random-like selection is performed. In the following a column selection strategy that ensures a concentrated form (2.73) of $R^{[r_{s,\ell}]}(x)$ is assumed.

Joint Optimization Problem

Given a mother code rate r and a sequence of ξ ascending relative target code lengths² $0 < n_1 < n_2 < \dots < n_\xi = 1$, a joint length-compatible optimization of shortening and puncturing similar to the one presented in Section 3.3.1 for the multi-rate case is proposed in the following. The same cost functions as introduced in Section 3.3.1 can be employed in general, where here the mean square gap to capacity is chosen. Thus, the objective of the optimization is to minimize the mean square gaps $\gamma_\ell|_{\text{dB}}$ of the ensemble thresholds $\tau_\ell|_{\text{dB}}$ to capacity at all effective relative code lengths n_ℓ with corresponding virtual code rate pairs $(r_{s,\ell}, r_{p,\ell})$ from (4.5) and (4.6), i.e.,

$$\begin{aligned} C(\gamma_1, \dots, \gamma_\xi) &= \frac{1}{\xi} \sum_{\ell=1}^{\xi} (\gamma_\ell|_{\text{dB}})^2 = \frac{1}{\xi} \sum_{\ell=1}^{\xi} (\tau_\ell|_{\text{dB}} - \mathbf{q}(r)|_{\text{dB}})^2 \\ &= \frac{1}{\xi} \left(\underbrace{\left(\tau(\Lambda, R)|_{\text{dB}} - \mathbf{q}(r)|_{\text{dB}} \right)^2}_{\text{mother code}} + \sum_{\ell=1}^{\xi-1} \underbrace{\left(\tau(\Lambda^{[r_{s,\ell}]}, R^{[r_{s,\ell}]}, \check{\Pi}^{[n_\ell]})|_{\text{dB}} - \mathbf{q}(r)|_{\text{dB}} \right)^2}_{\text{length-reduced codes}} \right). \end{aligned} \quad (4.9)$$

The optimization variables are, thus, the $2\xi - 1$ polynomials:

- $\Lambda(x) =: \Lambda^{[r_{s,\xi}]}(x)$: a variable node degree distribution for the mother code at the original code length (i.e., $n_\xi = 1$),
- $\Lambda^{[r_{s,\ell}]}(x)$: a shortened variable node degree distribution for each virtual shortened code rate $r_{s,\ell}$, with $\ell = 1, \dots, \xi - 1$,
- $\Pi^{[r_{p,\ell}]}(x)$: a puncturing polynomial for each virtual punctured code rate $r_{p,\ell}$, with $\ell = 1, \dots, \xi - 1$.

As mentioned above, all shortened check node degree distributions $R^{[r_{s,\ell}]}(x)$ are in concentrated form, which can later be ensured by the employed check matrix construction method.

²The original code length is always included as the highest one, i.e., $n_\xi = 1$.

To make the resulting multi-length ensemble length-compatible, constraints (3.42) to (3.46) from the multi-rate optimization in Section 3.3.1 are included in the proposed multi-length optimization problem, with appropriately adjusted indexing of the (virtual) code rates.

The proposed multi-length optimization problem is then given by

$$\min \quad \frac{1}{\xi} \sum_{\ell=1}^{\xi} (\gamma_{\ell}|_{\text{dB}})^2 \quad (4.10a)$$

$$\text{subject to} \quad \frac{\Lambda_i^{[r_{s,\ell}]}}{1-r_{s,\ell}} \stackrel{(3.42)}{\geq} \frac{\Lambda_i^{[r_{s,\ell-1}]}}{1-r_{s,\ell-1}}, \quad \ell = 2, \dots, \xi \quad (4.10b)$$

$$(1-r_{s,\ell-1}) \Lambda_i^{[r_{s,\ell}]} - (1-r_{s,\ell}) \Lambda_i^{[r_{s,\ell-1}]} \stackrel{(3.43)}{\leq} r_{s,\ell} - r_{s,\ell-1}, \quad \ell = 2, \dots, \xi \quad (4.10c)$$

$$\Pi_i^{[r_{p,1}]} \Lambda_i \stackrel{(3.44)}{\leq} \frac{1-r}{1-r_{s,1}} \Lambda_i^{[r_{s,1}]} \quad (4.10d)$$

$$\Pi_i^{[r_{p,\ell}]} \stackrel{(3.45)}{\leq} \Pi_i^{[r_{p,\ell-1}]}, \quad \ell = 2, \dots, \xi - 1 \quad (4.10e)$$

$$\sum_{i=2}^{d_v} \Pi_i^{[r_{p,\ell}]} \Lambda_i \stackrel{(3.46)}{=} 1 - \frac{r}{r_{p,\ell}} = 1 + n_{\ell} r - n_{\ell} - r, \quad \ell = 1, \dots, \xi - 1 \quad (4.10f)$$

$$0 \leq \Pi_i^{[r_{p,\ell}]} \leq 1, \quad \ell = 1, \dots, \xi - 1 \quad (4.10g)$$

$$\Lambda^{[r_{s,\ell}]}(1) = 1, \quad \Lambda_i^{[r_{s,\ell}]} \geq 0, \quad \ell = 1, \dots, \xi \quad (4.10h)$$

$$R^{[r_{s,\ell}]}(x) \text{ from (3.15)}, \quad \ell = 1, \dots, \xi \quad (4.10i)$$

$$\lambda_2 \stackrel{(2.50)}{<} \frac{e^{r\tau(\Lambda, R)}}{\sum_{j=2}^{d_c} \rho_j(j-1)} \quad (4.10j)$$

$$\lambda_2^{[r_{s,\ell}]} < \frac{e^{r\tau(\Lambda^{[r_{s,\ell}]}, R^{[r_{s,\ell}]}, \check{\Pi}^{[n_{\ell}]})}}{\sum_{j=2}^{d_c} \rho_j^{[r_{s,\ell}]}(j-1)}, \quad \ell = 1, \dots, \xi - 1 \quad (4.10k)$$

$$\Pi_2^{[r_{p,\ell}]} < \frac{1}{\lambda_2^{[r_{s,\ell}]} \sum_{j=2}^{d_c} \rho_j^{[r_{s,\ell}]}(j-1)}, \quad \ell = 1, \dots, \xi - 1. \quad (4.10l)$$

Constraints (4.10b) – (4.10i) are in accordance with the constraints of the multi-rate optimization (3.51) on page 85. A slight modification is required for the stability conditions (4.10j) – (4.10l), though. The original stability condition as introduced in (2.50) is only applicable to the mother code here and represented by (4.10j). The shortened degree distributions $\Lambda^{[r_{s,\ell}]}$ are not actually used at the respective shortened code rates $r_{s,\ell}$, but employed together with the puncturing polynomials $\Pi^{[r_{p,\ell}]}$ resulting in the original mother code rate r . In the stability condition (4.10k) for the shortened codes, the noise variance σ_n^2 is, thus, determined by the threshold τ

together with the mother code rate r instead of the shortened code rate $r_{s,\ell}$ by

$$\frac{1}{\sigma_n^2} = 2r\tau \left(\Lambda^{[r_{s,\ell}]}, R^{[r_{s,\ell}]}, \check{\Pi}^{[n_\ell]} \right). \quad (4.11)$$

Following the same argumentation, the puncturing stability condition (3.38) is included in constraint (4.10l) for all code rate pairs $(r_{s,\ell}, r_{p,\ell})$, for $\ell = 1, \dots, \xi - 1$, to account for the fact that puncturing is applied to the respective shortened code at each code length. For the multi-rate optimization, in contrast, the puncturing stability condition is only included for the mother code degree distribution in connection with the highest punctured code rate, since puncturing is always applied to the mother code.

This multi-length optimization problem can be solved in exactly the same way as the multi-rate optimization problem (3.51) using the Simulated Annealing (SA) method from Section 2.7.2 on pages 33 ff. To enable reliable optimization results for arbitrary ranges of code lengths, the DDE method with puncturing and $b = 8$ quantization bits is used to compute all thresholds. Moreover, the same two-phase approach is used that was introduced in Section 3.3.1 and is depicted in Figure 3.13.

4.1.3 Simulation Examples

To evaluate the performance of the proposed multi-length optimization method from the previous section, the following multi-length setups are considered:

- supported relative code lengths $n_\ell \in \{n_1, n_2, \dots, n_6\} = \{\frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}$,
- maximum variable node degree $d_v = 5$,
- concentrated check node degree distributions,
- code rates $r \in \{0.1, 0.2, \dots, 0.9\}$.

To this end, for each code rate r one separate multi-length ensemble is optimized by solving (4.10) using the SA approach.

Ensemble Thresholds

The gaps to capacity γ of the optimized ensemble thresholds for code rates 0.2, 0.5, and 0.9 are shown in dB over the relative code lengths n_ℓ in Figure 4.3 (solid lines).³ As reference, the dotted lines show the ensemble thresholds that are obtained if a dedicated degree distribution is optimized for each of the three code rates 0.2, 0.5, and 0.9 without any multi-length support. In this reference case, a dedicated code

³Even though the code length is reduced in this setup, it should be noted that the consideration of ensemble thresholds does not account for any finite length effects. That means that the asymptotic analysis assumes the mother code length as well as the reduced code length to both tend to infinity. For this reason the threshold values of the dedicated case are constant for all relative code lengths in Figure 4.3, whereas the penalty that is observed for the multi-length setups is caused solely by the asymptotic effects of concurrent shortening and puncturing.

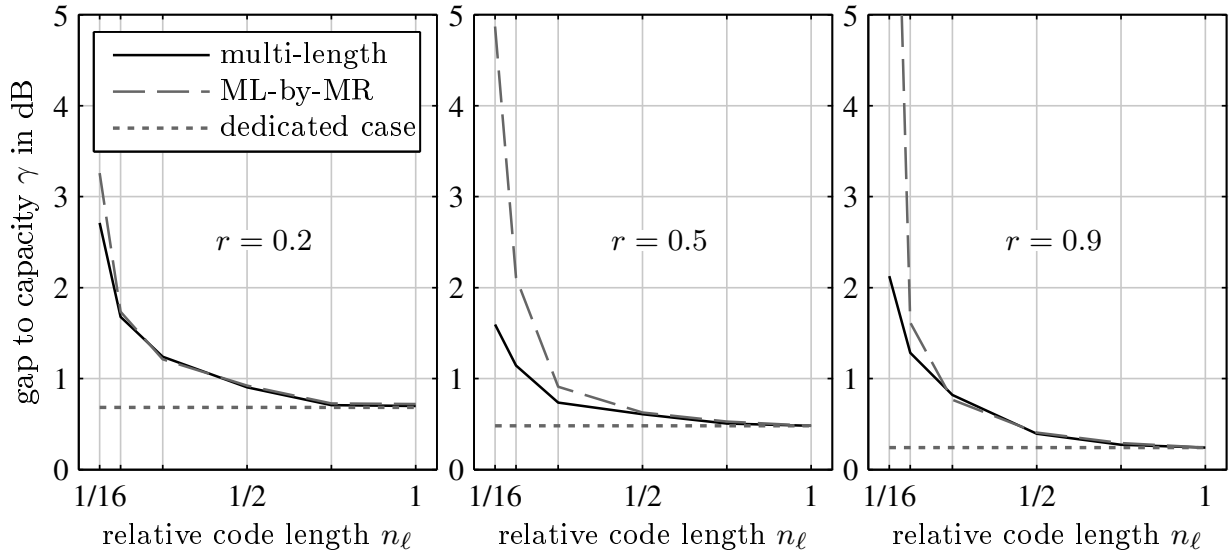


Figure 4.3: Gaps of ensemble thresholds to capacity over the relative reduced code length n_ℓ for different code rates r . Gaps are shown for the proposed multi-length optimization, the *multi-length by multi-rate* case, and the case of using a dedicated code for each code length.

needs to be constructed for each target code length, and since these codes can all use the same optimized degree distribution, the ensemble threshold is equal at all code lengths. Obviously, for all shown code rates, at relative code length $n_\ell = 1$ the ensemble thresholds of the multi-length setup are quasi identical to the dedicated case and increasingly deviate from it for smaller relative code lengths $n_\ell < 1$.

The question arises, whether it is possible to achieve similar multi-length performance by simply using results from the multi-rate optimization (3.51) from the previous chapter. By using the appropriate (virtual) code rates (4.5) and (4.6) as input to the multi-rate optimization, the same multi-length configuration can be obtained. The results of this optimization are shortened degree distributions with ensemble thresholds optimized for the shortened code rates $r_{s,\ell}$ and puncturing polynomials optimized for the punctured code rates $r_{p,\ell}$. Unfortunately, simply combining these *good* shortened degree distributions with the *good* puncturing polynomials does not nearly result in the same threshold performance as the proposed multi-length optimization. This setup is denoted *multi-length by multi-rate* (ML-by-MR) and results for this abuse of a multi-rate ensemble for length reduction are included in Figure 4.3 (dashed lines) for the same configuration as for the multi-length ensemble. It can be seen that the ML-by-MR setup yields almost identical ensemble thresholds as the multi-length setup for moderate reductions of code length ($n_\ell \geq \frac{1}{2}$). For very short relative code lengths, however, the thresholds obtained from the ML-by-MR optimization strongly deviate from those of the multi-length case by up to as much as 3 dB for $r = 0.5$ and $n_\ell = \frac{1}{16}$. This trend becomes more pronounced at large code rates and is less apparent at low code rates. For $r = 0.9$ and $n_\ell = \frac{1}{16}$ the ML-by-MR ensemble does not even converge to a valid ensemble threshold, as the puncturing stability condition is violated.

Altogether, the penalty of using a single mother code and reducing its code length

by puncturing and shortening compared to the case of using dedicated codes depends on the code rate and the minimum relative code length $n_\ell = n_{\min}$ that is supported. As a similar observation to the multi-rate case, this penalty is mainly caused by the inherent performance degradation of punctured codes. In contrast to the multi-rate case, where only small puncturing fractions have to be used if a high mother code rate is chosen, the mother code rate of the multi-length ensemble has to be equal to the target code rate r . The maximum puncturing fraction is, thus, fully determined by the minimum relative code length $n_{\min}$. The range of code lengths that can be achieved for a specific code rate depends on the penalty in ensemble threshold performance that can be tolerated as trade off for the reduced hardware complexity gained by using only a single mother code. To quantify this effect, the mean gap of the multi-length ensemble thresholds to that of the dedicated case (i.e., the mean distance between the solid and dotted lines in Figure 4.3) is computed for each code rate and denoted $\bar{\gamma}_{\text{ml}}|_{\text{dB}} - \gamma_{\text{ded}}|_{\text{dB}}$. By computing this measure for different minimum relative code lengths $n_{\min}$, a set of curves is obtained and depicted in Figure 4.4. The values of these curves answer the question: For a given code rate r and a given minimum relative code length $n_{\min}$, what is the average penalty in threshold performance of using the proposed length reduction to achieve relative code lengths between $n_{\min}$ and 1 compared to the case of using dedicated codes? As expected, the penalty becomes smaller for increasing minimum relative code lengths $n_{\min}$ at all code rates. Furthermore, for decreasing $n_{\min}$, the difference between the code rates becomes more pronounced, where code rates around 0.5 feature the smallest penalty and smaller code rates suffer from a larger penalty than larger code rates. For comparison, the x-markers⁴ show the mean gaps $\bar{\gamma}_{\text{mr}}|_{\text{dB}} - \gamma_{\text{ded}}|_{\text{dB}}$ of the ML-by-MR ensemble thresholds

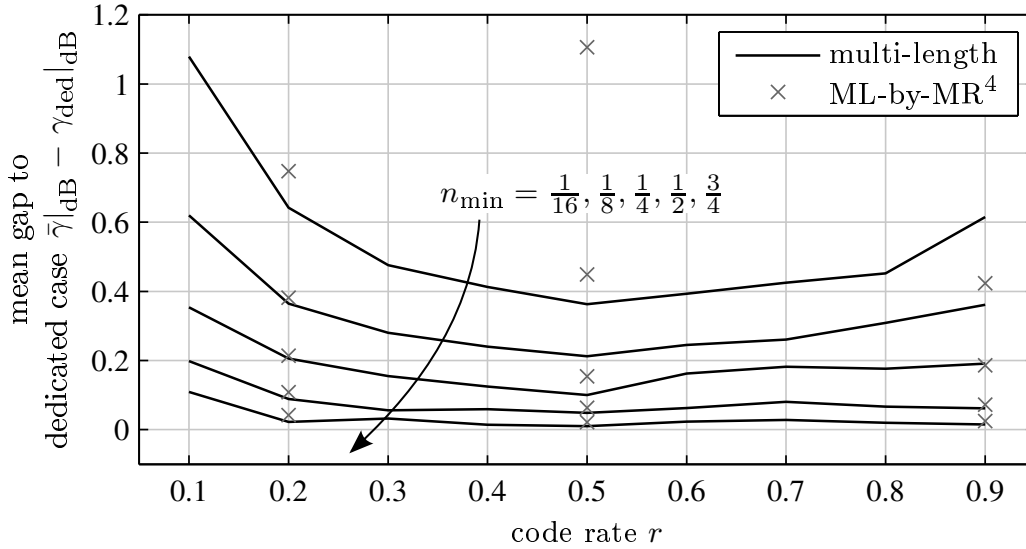


Figure 4.4: Mean gaps of the multi-length thresholds to the ones of the dedicated degree distributions over the code rate for different minimum relative code lengths $n_{\min}$. The x-markers show corresponding values that are achieved for the ML-by-MR optimization.

⁴There is no x-marker for $r = 0.9$ and $n_{\min} = 1/16$ as no valid threshold was found for this case due to a violation of the puncturing stability condition.

to that of the dedicated case (i.e., the mean distance between the dashed and dotted lines in Figure 4.3). This comparison confirms, that for a reduction of relative code length down to $n_{\min} = \frac{1}{2}$, results of the ML-by-MR optimization can be (ab)used for length adaptation with almost no average performance loss to the multi-length optimization. For $n_{\min} < \frac{1}{2}$, however, this average performance loss quickly increases and renders the ML-by-MR optimization unsuitable for length adaptation.

It should again be noted, that no multi-rate features are considered for any of the shown setups in this section. The scheme that is referred to as “multi-length by multi-rate” uses the proposed multi-rate optimization from the previous chapter, but is solely used for length reduction by concurrent shortening and puncturing here.

BER Performance

In addition to the asymptotic threshold results, BER simulations are conducted for the multi-length optimization and the dedicated case with BP decoding and $I_{\max} = 100$ decoding iterations. For each of the nine multi-length ensembles with code rates $r = 0.1, \dots, 0.9$, one mother code of length $N = 10080$ is constructed using the optimized matrix construction Algorithm 6 (MuRiPEG) on page 95. Even though this algorithm is originally intended to be used in the context of multi-rate ensembles, the finite length effects (check node concentration, recoverability of punctured bits, ACE properties) that were considered are equally applicable to the scenario of using concurrent shortening and puncturing for length reduction. The target code rate r is specified as mother code rate r_μ and the virtual code rates corresponding to the relative target code lengths $n_\ell \in \{\frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}$ are specified as effective shortened and punctured code rates as input to Algorithm 6. This construction method is referred to as *multi-length improved progressive edge-growth* (MuLiPEG) in the following. Additionally, 54 reference codes – one for each code rate r and each code length $n_\ell N \in \{630, 1260, \dots, 7560, 10080\}$ – are constructed with the dedicated optimized degree distributions from the previous section and the PEG algorithm.

In the upper plot of Figure 4.5 the BER over E_s/N_0 curves for the MuLiPEG method and the dedicated PEG codes are shown for code rates 0.2, 0.5, and 0.9. For MuLiPEG, curves are shown for effective relative code lengths $n_\ell \in \{\frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\}$, while for the dedicated PEG codes only curves for the largest and smallest relative code lengths $n_\ell \in \{\frac{1}{16}, 1\}$ are included. As predicted by the ensemble threshold results in Figure 4.3, at all three shown code rates in Figure 4.5 the MuLiPEG codes perform almost identical⁵ to the dedicated PEG codes at the maximum code length and suffer from a distinctly increasing performance degradation for decreasing code lengths. This behavior is highlighted by the lower plot of Figure 4.5, in which the gaps $\gamma^{[10^{-4}]}$ of the BER curves to capacity at BER 10^{-4} are shown in dB over the relative code length n_ℓ for both setups. Notably, at $r = 0.9$, the MuLiPEG codes perform much closer to the dedicated codes than predicted by the ensemble thresholds.

⁵Especially at low code rates, the mother codes constructed by the MuLiPEG algorithm even show better waterfall performance than the original PEG codes, which is caused by the different degree sequence ordering. The drawback of this improvement in the waterfall region is a degradation of the error floor behavior (cf. Section 3.3.2 and especially Figure 3.18).

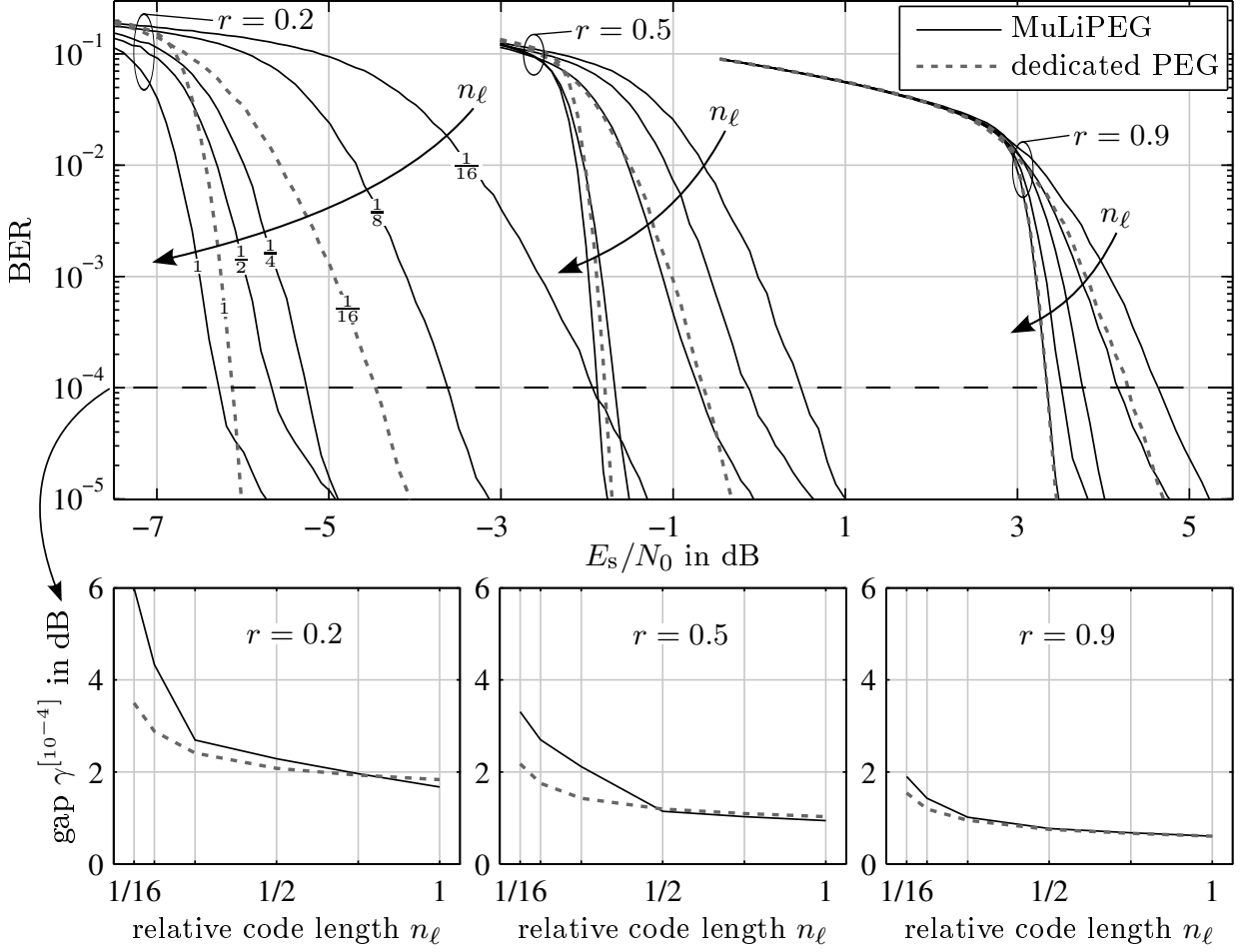


Figure 4.5: Top: BER over E_s/N_0 for $r \in \{0.2, 0.5, 0.9\}$. For MuLiPEG curves are shown for $n_\ell \in \{\frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\}$, for the dedicated PEG codes curves are shown only for $n_\ell \in \{\frac{1}{16}, 1\}$. Bottom: gaps $\gamma_{[10^{-4}]}^{\text{in dB}}$ to capacity at BER 10^{-4} over relative code length $n_\ell \in \{\frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\}$ for both setups.

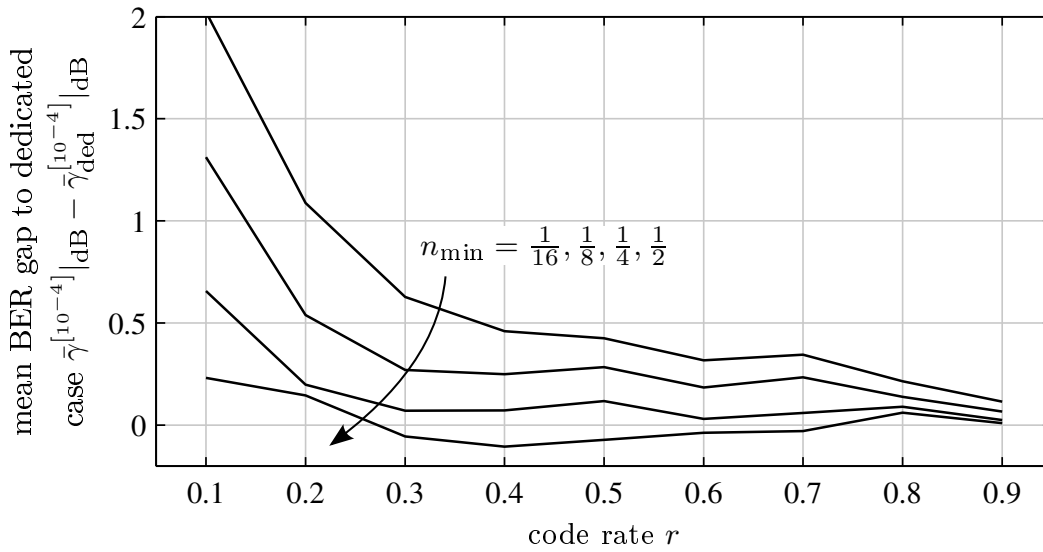


Figure 4.6: Mean gaps $\bar{\gamma}_{[10^{-4}]}^{\text{in dB}} - \bar{\gamma}_{\text{ded}}^{\text{in dB}}$ at BER 10^{-4} of MuLiPEG to the dedicated case over the code rate for different minimum relative code lengths $n_{\min}$.

Equivalently to the illustration for ensemble thresholds in Figure 4.4, an additional representation of all BER results is provided in Figure 4.6 to answer the question: For a given code rate r and a given minimum relative code length $n_{\min}$, what is the average penalty in BER performance of using the proposed length reduction to achieve relative code lengths between $n_{\min}$ and 1 compared to the case of using dedicated PEG codes? The mean gap $\bar{\gamma}^{[10^{-4}]}|_{\text{dB}} - \bar{\gamma}_{\text{ded}}^{[10^{-4}]}|_{\text{dB}}$ of the BER curves to the dedicated case at BER 10^{-4} is shown for minimum relative code lengths $n_{\min} \in \{\frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}\}$. There are two notable differences to the ensemble threshold predictions from Figure 4.4. First, for small values of $n_{\min}$ the average BER penalty (almost) monotonically decreases with increasing rate, whereas the average penalty in ensemble threshold exhibits a minimum at $r = 0.5$ in Figure 4.4. Second, for $n_{\min} = \frac{1}{2}$ the BER penalty even takes negative values at some code rates, which is explained by the fact that the mother codes constructed by the MuLiPEG algorithm in some cases show better waterfall behavior than the original PEG codes due to the different degree sequence ordering.

4.1.4 Joint Multi-Length and Multi-Rate Considerations

Due to the similarity between the multi-rate optimization presented in Section 3.3 and the multi-length optimization presented here, a straightforward generalization to the case of jointly achieving multi-rate and multi-length capabilities by shortening and puncturing is possible. In the multi-rate optimization only the application of either shortening or puncturing at any moment is considered. The multi-length optimization, on the other hand, allows concurrent shortening and puncturing but is restricted to fixed (virtual) code rate pairs to reduce the code length while keeping the code rate unchanged. By allowing concurrent shortening and puncturing using arbitrary virtual code rate pairs, a simultaneous adaptation of the code rate and the code length can be achieved. For the exemplary case of mother code rate $r = 0.5$ and (virtual) code rates from 0.1 to 0.9, the achievable combinations of code rates and lengths for these three methods are depicted in a “length-rate-plane” in Figure 4.7. Allowing concurrent shortening and puncturing with arbitrary combinations of the pre-defined (virtual) code rates results in an increased variety of length and rate combinations. However, to ensure good performance, an explicit optimization problem has to be formulated that takes into account all of these combinations. The derivation of this problem and its solution directly follows the multi-length optimization problem (4.10), with the only difference being that all combinations of shortened and punctured code rates need to be considered instead of just some pre-defined fixed pairs. In the example of Figure 4.7, e.g., the cost function has to compute the mean square gap to capacity of 25 ensembles instead of just 5 for the pure multi-length case or 9 for the pure multi-rate case.

This approach can be taken even further, by allowing a “smooth” adaptation of code rates and lengths to reach arbitrary points in the gray shaded area in Figure 4.7. Instead of having to optimize dedicated degree distributions and puncturing polynomials for each of the additionally achievable points, however, a simple interpolation of the coefficients can be employed. This is achieved as follows. By choosing an arbitrary combination of n_{ℓ} and r_{eff} from the gray shaded area, the resulting shortening

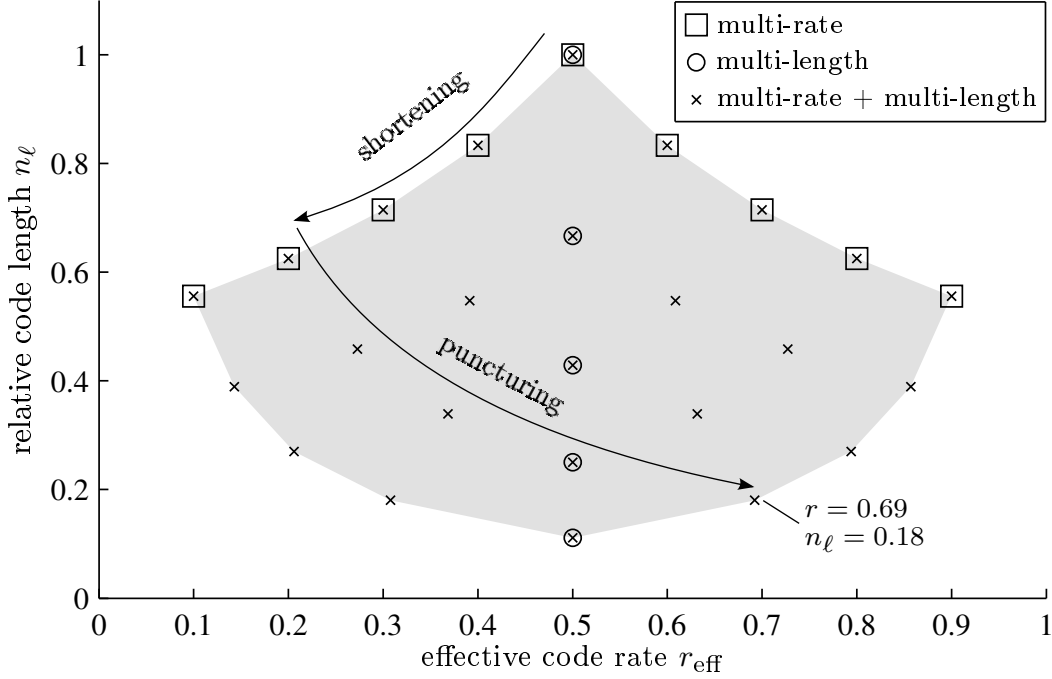


Figure 4.7: Length-rate-plane with combinations achievable by shortening and puncturing according to the multi-rate scheme from Section 3.3, the multi-length scheme proposed in this section, and a joint consideration of multi-rate and multi-length capabilities.

and puncturing fractions k_s and m_p can be determined by (4.2) and (4.3) on page 105. These values in turn define the virtual shortened and punctured code rates r_s and r_p according to (3.1) on page 51 and (3.20) on page 67, respectively. The shortened degree distribution for the virtual shortened code rate r_s is then obtained by linear interpolation of the coefficients of the pre-optimized degree distributions $\Lambda^{[r_i]}$ and $\Lambda^{[r_{i+1}]}$ of the two neighboring code rates. Equally, the puncturing polynomial for the virtual punctured code rate r_p is obtained by linear interpolation of the coefficients of the pre-optimized puncturing polynomials $\Pi^{[r_j]}$ and $\Pi^{[r_{j+1}]}$ of the two neighboring code rates. As optimized degree distributions and puncturing polynomials for neighboring code rates obey a certain continuity, the interpolated values at (virtual) code rates in between will be very close to the optimal ones as long as the (virtual) code rates are not too far apart.

This joint multi-rate and multi-length scheme offers a tremendous amount of flexibility in terms of supported code rates and lengths. However, for cases in which large fractions of the parity bits are punctured, it suffers the same performance degradation as the pure multi-rate and multi-length approaches. Still, if only moderate ranges of code rates and lengths need to be supported, concurrent shortening and puncturing offers performance that is competitive to the case of using one dedicated code per combination of code rate and length but requires only a single encoder and decoder implementation of the mother code that can be used for all setups without any additional complexity.

4.1.5 Concluding Remarks: MuLiPEG

Concurrently shortening and puncturing a mother code word is an effective method of very low-complexity to flexibly reduce the code length using only a single LDPC mother encoder and decoder. No other method is currently known that provides the same flexibility and is generally applicable to LDPC codes.⁶ Furthermore, a straightforward combination of the proposed MuLiPEG method with the multi-rate capabilities presented in Section 3.3 is possible.

The investigations in this section show that a special multi-length optimization is superior to simply using optimized shortening and puncturing schemes. The BER simulations for a mother code length of $N = 10080$ confirm, that for high code rates the average performance loss of the proposed scheme to the case of using one dedicated code per code length is very small even if the code length is reduced to only $\frac{N}{16} = 630$, e.g., about 0.11 dB average penalty for $r = 0.9$ and about 0.21 dB average penalty for $r = 0.8$. For code rates below 0.5, however, this penalty quickly increases if similarly low relative code lengths have to be achieved.

4.2 Multi-Length Quasi-Cyclic LDPC Codes

The structure of QC LDPC codes inherently allows the construction of codes with different lengths from one mother lifting matrix $\mathbf{A}$ of fixed dimensions $M' \times N'$. As described in Section 2.9, the parity check matrix $\mathbf{H}^{[Z]}$ of a QC LDPC code is obtained by replacing each entry $A_{m',n'}$ of the lifting matrix by the $Z \times Z$ matrix $\mathbf{I}_s^{A_{m',n'}}$ (which is either a cyclically shifted identity matrix or an all-zero matrix). The resulting code length is given by $N = ZN'$, while the code rate $r = \frac{ZN' - ZM'}{ZN'} = \frac{N' - M'}{N'}$ is independent of the lifting factor Z . By using different values for Z , codes with fixed code rate and different code lengths are obtained from the mother lifting matrix $\mathbf{A}$.

4.2.1 Encoding and Decoding

To qualify as a multi-length LDPC code, encoder and decoder have to be able to process the codes of all resulting lengths based on only one common implementation for the mother lifting matrix. It can be shown that this constraint is fulfilled for most typical hardware architectures used in connection with QC LDPC codes.

Most QC LDPC decoder architectures follow either the *circulant perspective* or the *protograph perspective* [RN03, Zha07, JDA⁺07, RL09] which are illustrated in Figure 4.8. Both of these start from the protograph representation of the lifting matrix as introduced in Section 2.9 on page 42.

⁶As mentioned above, the approach of [KCY12] also uses concurrent shortening and puncturing to reduce the code length. This approach, however, is only applicable to IRA codes – a special subclass of LDPC codes – which is why no fair comparison to the proposed method is possible.

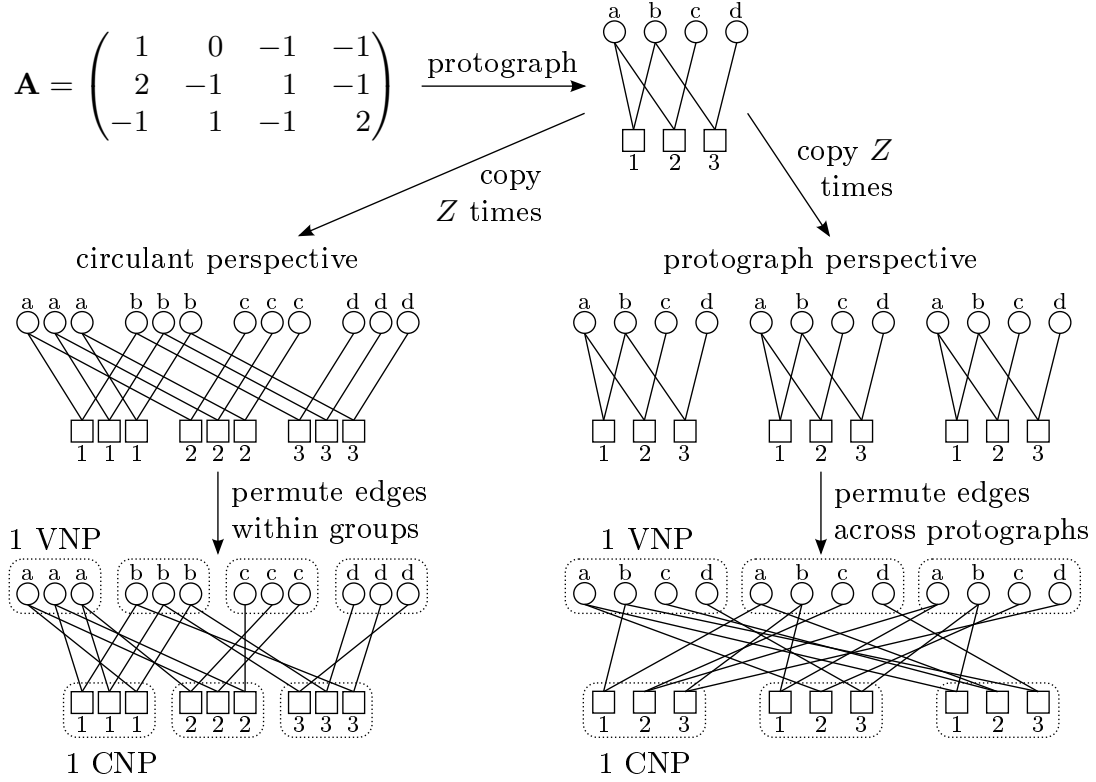


Figure 4.8: Construction of a code's Tanner graph with $N = ZN' = 12$ variable and $M = ZM' = 9$ check nodes from the lifting matrix of size $M' \times N' = 3 \times 4$ with lifting factor $Z = 3$ by a copy-and-permute operation using the circulant perspective (left) and the protograph perspective (right). Each of the dotted boxes represents one variable (VNP) or check node processor (CNP) that is employed by an according hardware architecture.

In the circulant perspective each node of this protograph is copied Z times resulting in groups of nodes of the same type. Then, the edges between each pair of groups are cyclically permuted according to the entry in $\mathbf{A}$ that connects the two types. E.g., in Figure 4.8 before permutation the first check node of type “2” is connected to the first variable node of type “a”, and after cyclic permutation by two position (according to the lifting matrix entry $A_{2,1} = 2$) it is connected to the third variable node of type “a”. The Tanner graph that is constructed this way corresponds to the check matrix $\mathbf{H}^{[Z]}$ that is obtained from $\mathbf{A}$ as described in Section 2.9. A decoder that is based on this representation, is composed of N' variable node processing units and M' check node processing units, one for each type (or group) of nodes. These units are connected by memory elements according to the lifting matrix entries, each of which stores Z extrinsic messages that are exchanged between Z variable and check nodes of the respective group. Each processing unit can only compute one variable-to-check or check-to-variable message at a time, which is why the Z nodes in each group are processed sequentially by the processing unit. The only requirement this architecture needs to fulfill to be compatible with using multiple lifting factors $Z_1, \dots, Z_{\max}$ are memory elements that can store the maximum number of $Z_{\max}$ messages. If $Z < Z_{\max}$ is used, some of the memory locations are simply not addressed, and the next BP

decoding iteration starts as soon as each processing unit has computed Z messages sequentially.

The protograph perspective, on the other hand, starts from the same protograph but considers the initial Z -fold copying as a Z -fold duplication of that whole protograph. The subsequent permutation can then be seen as spreading the edges across the different protographs, again according to the corresponding entries in the lifting matrix $\mathbf{A}$. Except for a different order of rows and columns, this representation results in the same check matrix as for the circulant perspective. A decoder following this architectural design perspective consists of $Z_{\max}$ identical constituent decoders, each with one variable node and one check node unit. The i -th constituent decoder updates all variable and all check nodes within the i -th copy of the protograph sequentially. This can be seen as one large *vector processor* composed of $Z_{\max}$ identical constituent variable and check node processors [RN03]. This architecture also directly supports multiple values of Z by simply deactivating a number of processors for $Z < Z_{\max}$. The massively parallelized implementation of the layered BP decoder presented in Section 6.3 is an example for an architecture in the protograph perspective.

For the encoding of QC LDPC codes, several methods have been proposed that are based only on the lifting matrix $\mathbf{A}$ and exploit the quasi-cyclic property by using Z bit wide feedback shift registers [LCZ⁺06, RL09]. To employ such a decoder for varying values of Z , the shift registers need to be of width $Z_{\max}$, and the position of the feedback tap needs to be adaptable to enable cyclic shifting for values $Z < Z_{\max}$.

Altogether, for one fixed mother lifting matrix, a multi-length QC LDPC code can be realized by expanding this lifting matrix with different lifting factors Z as long as a suitable encoder and decoder architecture is employed.

4.2.2 Jointly Optimized Lifting Matrix Construction

For the design of a good multi-length QC LDPC code, a lifting matrix $\mathbf{A}$ needs to be found that delivers good performance for multiple lifting factors Z . The expanded Tanner graphs for all values of Z are constructed from the same protograph and thus share the same degree distribution and the same local structure, leading to the same asymptotic performance behavior in terms of their density evolution thresholds. Consequently, asymptotic analysis and optimization solely needs to be done for the lifting matrix and no specific multi-length properties have to be considered. It is, thus, sufficient to use degree distribution pairs optimized for the case of fixed code rate and fixed code length (cf. Section 2.7) for the construction of the lifting matrix.

While the asymptotic behavior is the same for all lifting factors Z , the exact placement and values of the non-negative entries in the lifting matrix $\mathbf{A}$ distinctively influences the finite length performance of the different expanded codes. In the following, an optimized lifting matrix construction method is developed that jointly considers multiple lifting factors $Z_{\min} = Z_1 < Z_2 < \dots < Z_\xi = Z_{\max}$. The dimension $M' \times N'$ and the variable node degree distribution $\Lambda(x)$ are assumed to be given as input. As the proposed method is based on an extension of the PEG algorithm to the case of

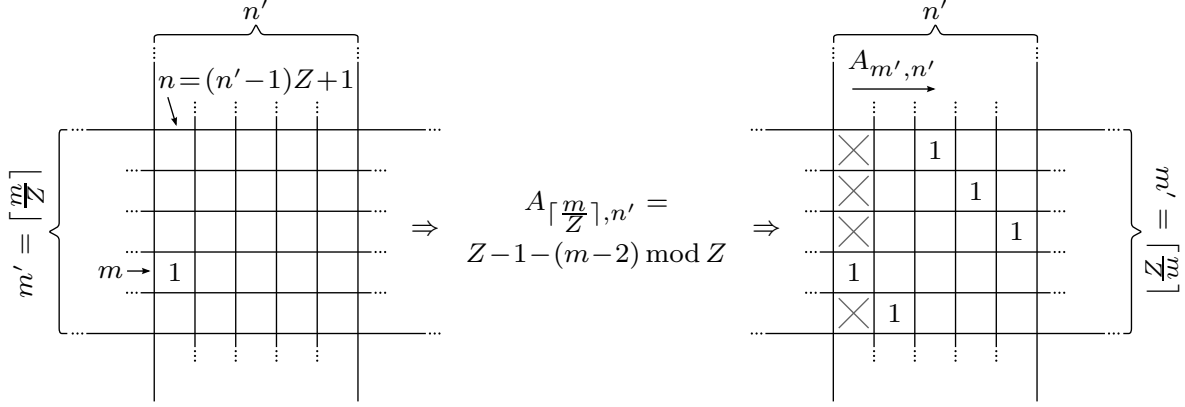


Figure 4.9: Visualization of the circulant constraint. If a 1 is placed at row m and column n in the expanded check matrix (left), the remaining entries in this group of variable and check nodes have to form an identity matrix, cyclically shifted by $Z - 1 - (m - 2) \bmod Z$ positions.

QC LDPC codes as presented in [LK04], an explanation of how to construct a lifting matrix using the PEG algorithm is first provided.

PEG for QC LDPC Codes [LK04]

The PEG algorithm can easily be constrained to yield QC LDPC codes as follows. In the (initially empty) expanded check matrix, for each group of Z variable nodes corresponding to one column of the lifting matrix, only the first variable node is considered and connected to a check node chosen by the PEG method. To obtain a QC LDPC code, after this choice for the initial variable node, the remaining variable nodes in the same group are connected to those check nodes that result in a cyclically shifted identity matrix in the current variable and check node group. This dependency of nodes within one group is referred to as *circulant constraint* and is illustrated in Figure 4.9. Relations between node indices in the expanded check matrix and group indices and shift values in the lifting matrix can easily be derived:

- The first index in (variable or check node) group i' is given by $i = (i' - 1)Z + 1$.
- The last index in (variable or check node) group i' is given by $i = i'Z$.
- Given an arbitrary (variable or check node) index i , the corresponding group index is given by $i' = \lceil \frac{i}{Z} \rceil$, where $\lceil \cdot \rceil$ denotes rounding up.
- If the first variable node $n = (n' - 1)Z + 1$ in group n' is connected to any check node m , the shift value of the corresponding lifting matrix entry is given by

$$A_{\lceil \frac{m}{Z} \rceil, n'} = s^{[Z]}(m) := Z - 1 - (m - 2) \bmod Z, \quad (4.12)$$

with $(-1) \bmod Z := Z - 1$.

Furthermore, whenever a check node is connected to the current variable node, all check nodes from the same group have to be excluded from the set of candidates that may be chosen for further connections to the same variable node. If all connections have been placed at the first variable node of a group, the whole group is finished and the processing continues at the first variable node of the next group.

Proposed Multi-Length Quasi-Cyclic PEG (MuLQPEG) Algorithm⁷

Usually, the shift values $A_{m',n'}$ in the lifting matrix are restricted to the range 0 to $Z - 1$, as this is sufficient to describe all Z possible cyclically shifted identity matrices $\mathbf{I}_s^{A_{m',n'}}$. In the case of multiple lifting factors, as considered here, shift values in the range 0 to $Z_{\max} - 1$ are allowed. This yields the maximum degree of freedom for the largest lifting factor and at the same time does not restrict any of the smaller lifting factors, since a cyclic shift of a $Z_k \times Z_k$ matrix by $s > Z_k - 1$ positions can be equivalently described as a cyclic shift by $\tilde{s} = s \bmod Z_k$ positions.

Algorithm 7 Multi-Length Quasi-Cyclic PEG (MuLQPEG)

```

1: Input:  $M', N'; Z_1, \dots, Z_\xi; \mathbf{d}_v = (d_{v,1}, \dots, d_{v,N'})$ 
2: Initialize  $A_{m',n'} = -1, \forall m', n'$ 
3: Initialize check node group degrees  $\mathbf{d}_c = (d_{c,1}, \dots, d_{c,M'}) = (0, \dots, 0)$ 
4: for  $n' = 1, \dots, N'$  do
5:   for  $d = 1, \dots, d_{v,n'}$  do
6:     if  $d = 1$  then
7:       Choose  $m' \in \arg \min_{\forall m''} d_{c,m''}$  and  $A_{m',n'} \in \{0, \dots, Z_\xi - 1\}$  randomly
8:       Update  $d_{c,m'} = d_{c,m'} + 1$ 
9:     else
10:       $\mathcal{K}^{[Z_1]} = \bigcup_{\forall m': A_{m',n'} = -1} \{(m' - 1)Z_1 + 1, \dots, m'Z_1\}$ 
11:      for  $i = 1, \dots, \xi$  do
12:        Initialize  $\omega_m = \infty, \forall m$ 
13:        Using  $\mathbf{H}^{[Z_i]}$  and starting at variable node  $n = (n' - 1)Z_i + 1$ , expand
        a tree until no more nodes can be added, setting  $\omega_m = j$  for all check
        nodes  $m$  that are reached for the first time at depth  $j$ 
14:         $\mathcal{K}_{\text{girth}}^{[Z_i]} = \arg \max_{\forall m \in \mathcal{K}^{[Z_i]}} \omega_m$ 
15:         $\mathcal{K}_{\text{weight}}^{[Z_i]} = \arg \min_{\forall m \in \mathcal{K}_{\text{girth}}^{[Z_i]}} d_{c,\lceil m/Z_i \rceil}$ 
16:        if  $i < \xi$  then
17:           $\mathcal{K}^{[Z_{i+1}]} = \{m \in \{1, \dots, Z_{i+1}M'\} : \forall j \leq i, \exists \tilde{m} \in \mathcal{K}_{\text{weight}}^{[Z_j]}, \lceil \frac{m}{Z_{i+1}} \rceil = \lceil \frac{\tilde{m}}{Z_j} \rceil$ 
18:             $\wedge s^{[Z_{i+1}]}(m) \bmod Z_j = s^{[Z_j]}(\tilde{m})\}$ 
19:        end if
20:      end for
21:      Choose  $m \in \mathcal{K}_{\text{weight}}^{[Z_\xi]}$  randomly
22:       $A_{\lceil m/Z_\xi \rceil, n'} = s^{[Z_\xi]}(m)$  and  $d_{c,\lceil m/Z_\xi \rceil} = d_{c,\lceil m/Z_\xi \rceil} + 1$ 
23:    end if
24:  end for

```

⁷The proposed jointly optimized lifting matrix description follows the description in the Master's thesis [Wic14].

The proposed *multi-length quasi-cyclic PEG* (MuLQPEG) algorithm is fully described in Algorithm 7 and operates as follows. In each step of the algorithm and for each lifting factor Z_i , the expanded check matrix that corresponds to the current lifting matrix $\mathbf{A}$ is denoted $\mathbf{H}^{[Z_i]}$. The variable node group degree sequence $\mathbf{d}_v$ is computed from the given variable node degree distribution in ascending order by (2.85). Processing starts with an empty lifting matrix $\mathbf{A}$ (i.e., a matrix with all elements equal to -1) and all check node group degrees $d_{c,m'}$ set to zero. In a group-by-group manner, for each variable node group n' (column in the lifting matrix) $d_{v,n'}$ check node groups m' (rows in the lifting matrix) and associated shift values $A_{m',n'}$ are determined. For the first connection of each group, a random check node group of currently lowest degree $d_{c,m'}$ and a random shift value between 0 and $Z_\xi - 1 = Z_{\max} - 1$ is chosen. For the remaining connections, a set $\mathcal{K}^{[Z_i]}$ of check node candidates (i.e., rows in the expanded check matrix) is composed by successively considering all expanded check matrices $\mathbf{H}^{[Z_i]}$. The initial set $\mathcal{K}^{[Z_1]}$ contains the check nodes $(m' - 1)Z_1 + 1, \dots, m'Z_1$ for each check node group m' that is not (yet) connected to the current variable node group n' . These candidates represent the combinations of all unconnected check node groups and all shift values between 0 and $Z_1 - 1$. For all lifting factors Z_i the following three steps are then performed:

- Compose $\mathcal{K}_{\text{girth}}^{[Z_i]}$ as all check node candidates from $\mathcal{K}^{[Z_i]}$ that are unreachable or otherwise have maximum distance from the first variable node $n = (n' - 1)Z_i + 1$ in the current group n' .
- Compose $\mathcal{K}_{\text{weight}}^{[Z_i]}$ as all check node candidates m from $\mathcal{K}_{\text{girth}}^{[Z_i]}$ that belong to a group $m' = \lceil \frac{m}{Z_i} \rceil$ with minimum current degree $d_{c,m'}$.
- If $i < \xi$, compose $\mathcal{K}^{[Z_{i+1}]}$ as all check nodes $m \in \{1, \dots, Z_{i+1}M'\}$ for which for all smaller lifting factors Z_j , $j \leq i$, a check node $\tilde{m}$ exists in $\mathcal{K}_{\text{weight}}^{[Z_j]}$ such that m and Z_{i+1} result in the same group index as $\tilde{m}$ and Z_j and the shift value $s^{[Z_{i+1}]}(m)$, if taken modulo Z_j , is equal to the shift value $s^{[Z_j]}(\tilde{m})$.

At the smallest lifting factor Z_1 , check node candidates are simply chosen according to the PEG rule of maximizing the girth and uniformly growing row weights. The key element of the algorithm is the successive extension of the candidate set before any node is actually chosen. The candidates $\mathcal{K}^{[Z_{i+1}]}$ contain all check nodes in $\mathbf{H}^{[Z_{i+1}]}$ such that the group indices and shift values are still equivalent with respect to all smaller lifting factors Z_j , with $j \leq i$. An example for this extension of candidate sets within one check node group for lifting factors $Z_1 = 3$, $Z_2 = 4$, and $Z_3 = 7$ is illustrated in Figure 4.10.

After the candidate set $\mathcal{K}_{\text{weight}}^{[Z_\xi]}$ for the largest lifting factor has been found, one check node candidate m is randomly chosen resulting in the lifting matrix entry $A_{\lceil m/Z_\xi \rceil, n'} = s^{[Z_\xi]}(m)$. For each variable node group, this process is repeated until the desired number of connections is found, before processing continues with the next variable node group.

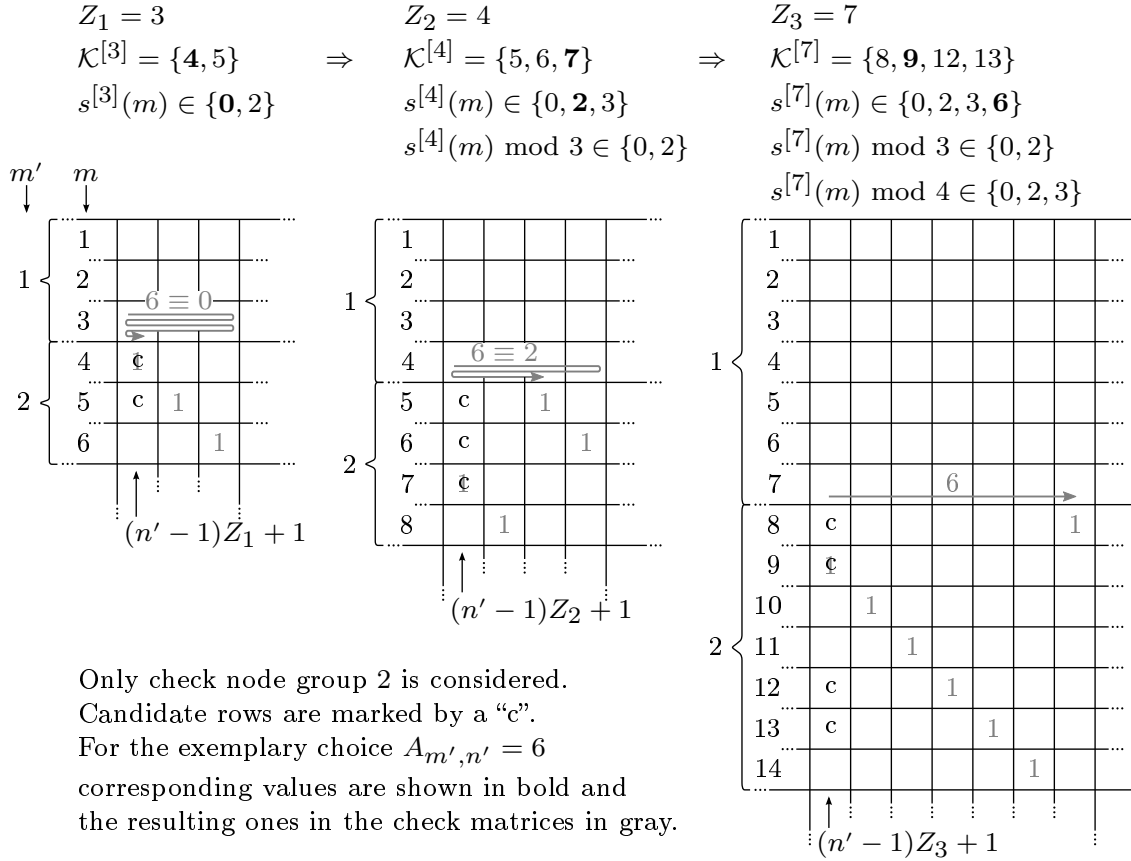


Figure 4.10: Exemplary extension of the candidate set $\mathcal{K}^{[Z_i]}$ from $Z_1 = 3$ to $Z_2 = 4$ and $Z_3 = 7$ for check node group $m' = 2$ only.

4.2.3 Simulation Examples

Code lengths 630 to 10080, code rate 1/2

As a first proof of concept, the proposed MuLQPEG method is compared to three reference schemes for code lengths $N \in \{630, 1260, 2520, 5040, 7560, 10080\}$ and code rate $r = 0.5$, achieved with a lifting matrix size of 35×70 and lifting factors $Z \in \{9, 18, 36, 72, 108, 144\}$.

MuLQPEG : A jointly optimized lifting matrix is constructed with Algorithm 7 and expanded to all target code lengths.

Dumb modulo : A lifting matrix is constructed with the conventional QC PEG algorithm [LK04] based on only the largest lifting factor $Z_{\max} = 144$ (i.e., entries are in the range $0, \dots, 143$). The resulting mother lifting matrix is expanded by all values $Z \in \{9, 18, 36, 72, 108, 144\}$ to obtain the target code lengths, where all shift values s larger than the current value of Z result in the equivalent shift by $s \bmod Z$ positions.

Dumb expand : A lifting matrix is constructed with the conventional QC PEG algorithm [LK04] based on only the smallest lifting factor $Z_{\min} = 9$ (i.e., entries are in the range $0, \dots, 8$). The resulting mother lifting matrix is simply expanded by all values $Z \in \{9, 18, 36, 72, 108, 144\}$ to obtain the target code lengths.

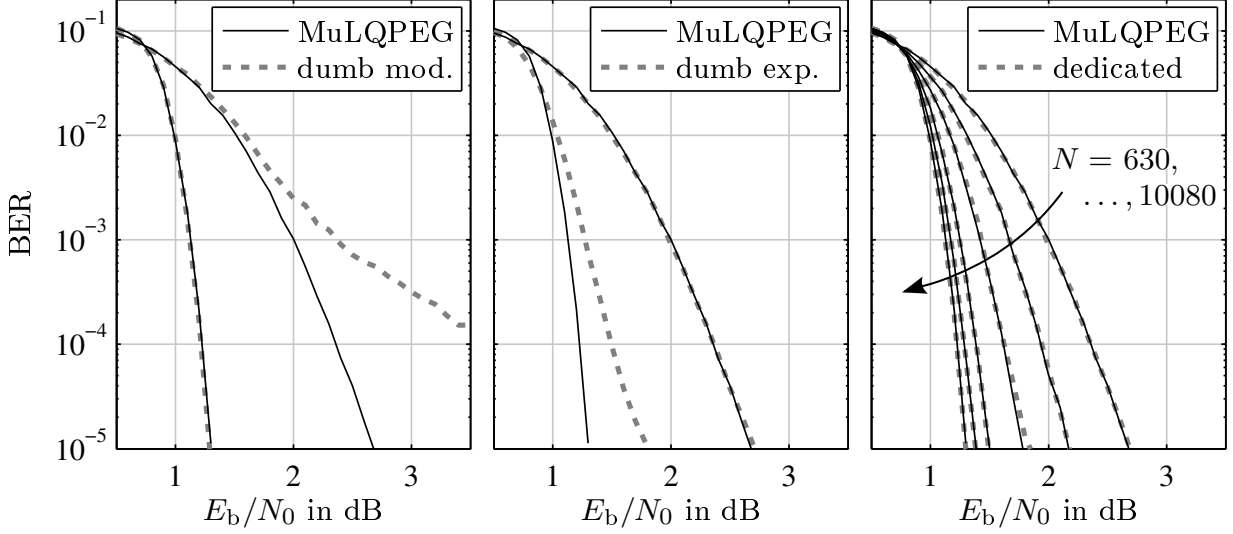


Figure 4.11: BER curves of the proposed MuLQPEG multi-length code compared to *dumb modulo* (left) and *dumb expand* (center) for $N = 10080$ and $N = 630$ as well as to the case of dedicated codes (right) for code lengths $N \in \{630, 1260, 2520, 5040, 7560, 10080\}$.

Dedicated : A dedicated QC LDPC code without any multi-length consideration is constructed for each code length N with the conventional QC PEG algorithm [LK04].

In all cases, the employed degree distribution with $d_v = 5$ is the same as the one that was used for the dedicated reference codes in Section 4.1.

In Figure 4.11, BER over E_b/N_0 curves after BP decoding with a maximum of $I_{\max} = 100$ iterations are shown. For the largest and smallest code lengths $N = 10080$ and $N = 630$, the proposed *MuLQPEG* multi-length code is compared to the *dumb modulo* scheme in the left plot and to the *dumb expand* scheme in the center plot. In the right plot, curves are shown for all code lengths N for *MuLQPEG* and the *dedicated* case. It is clearly visible, that the *dumb modulo* scheme performs equal to *MuLQPEG* for $N = 10080$ and much worse for $N = 630$, while the *dumb expand* scheme shows equal performance to *MuLQPEG* for $N = 630$ and is inferior at $N = 10080$. This agrees with the design concept of the two schemes, as the lifting matrix is optimized for the largest lifting factor by the *dumb modulo* scheme and for the smallest one by the *dumb expand* scheme. In these cases, using different lifting factors potentially creates additional cycles, which might have been avoidable, had these lifting factors been considered during lifting matrix construction. The comparison of *MuLQPEG* to the dedicated case even reveals that for this setup using only one multi-length mother code does not inflict any penalty in BER performance as compared to using one dedicated code per code length.

Code lengths 140 to 32760, code rates 0.2, 0.5, and 0.9

To even further explore the potential of the proposed MuLQPEG method, lifting matrices of width $N' = 70$ are constructed for code rates $r = 0.2$, $r = 0.5$, and $r = 0.9$

that are optimized for lifting factors $Z \in \{2, 4, 9, 18, 36, 72, 108, 144, 288, 378, 468\}$ resulting in achievable code lengths between $N = 140$ and $N = 32760$. In Figure 4.12 BER results are shown for MuLQPEG and again for the reference case of one dedicated QC LDPC code per code length. In the upper plot, the BER over E_s/N_0 is shown for the four code lengths $N = 140$, $N = 630$, $N = 10080$, and $N = 32760$. The three lower plots show the gap to capacity at BER 10^{-4} over the code length N . Apparently, even for this increased code length range and also for different code rates, nearly no performance penalty is inflicted by the use of only one jointly optimized MuLQPEG mother lifting matrix compared to using dedicated codes for each code length.

Additionally, in Figure 4.13 the proposed MuLQPEG scheme is compared to results for the MuLiPEG scheme from Section 4.1 (which achieves length reduction by concurrent shortening and puncturing) and to two approaches from the literature:

[MY05] : This method takes a lifting matrix for lifting factor Z as input and outputs a modified lifting matrix for a pre-defined lifting factor $c \cdot Z$, where c is an integer greater than one. The modified lifting matrix is constructed to result in as few short cycles as possible when expanded with $c \cdot Z$, while yielding the same expanded check matrix for the original lifting factor Z . By applying this method repeatedly, a multi-length QC LDPC code can be constructed, in which each code length must be an integer multiple of the code length directly below. The initial lifting matrix of width $N' = 70$ is constructed for $Z_1 = 3$ in the same way as for the MuLQPEG method (with the same optimized degree distribution and the QC PEG algorithm [LK04]). This lifting matrix is then successively modified to be used for lifting factors $Z \in \{9, 18, 36, 72, 144, 288\}$ resulting in code lengths from $N = 210$ to $N = 20160$.

[KCY12] : A special method to achieve multiple lengths by concurrent shortening and puncturing of IRA codes. The code rate and the supported code lengths can be arbitrarily chosen. This method is used to construct one IRA code of length $N = 32760$ for each code rate $r = 0.2$, $r = 0.5$, and $r = 0.9$. For $r = 0.5$ and $r = 0.9$ the same optimized degree distributions as for the MuLQPEG scheme are used. For $r = 0.2$, however, a separately optimized degree distribution has to be used since the number of degree 2 variable nodes needs to be at least M for IRA codes, which is not fulfilled by the degree distribution used for MuLQPEG. The code length of each of the three IRA codes is reduced to values $N \in \{26460, 20160, 10080, 7560, 5040, 2520, 1260, 630, 450, 280, 140\}$.

The curves for the MuLQPEG scheme in Figure 4.13 are the same as in Figure 4.12 and are nearly identical to the case of dedicated QC LDPC codes which are not again shown here. At its smallest code length $N = 210$, the method of [MY05] slightly outperforms MuLQPEG as it is based on an unconstrained optimized initial lifting matrix at the smallest code length, while MuLQPEG uses a jointly optimized lifting matrix. However, for increasing code lengths the method of [MY05] is not capable of keeping up with the performance of MuLQPEG and is the only scheme that for all three code rates reaches a point where it does not gain from further increasing the code length. The IRA codes constructed by the method of [KCY12] show very

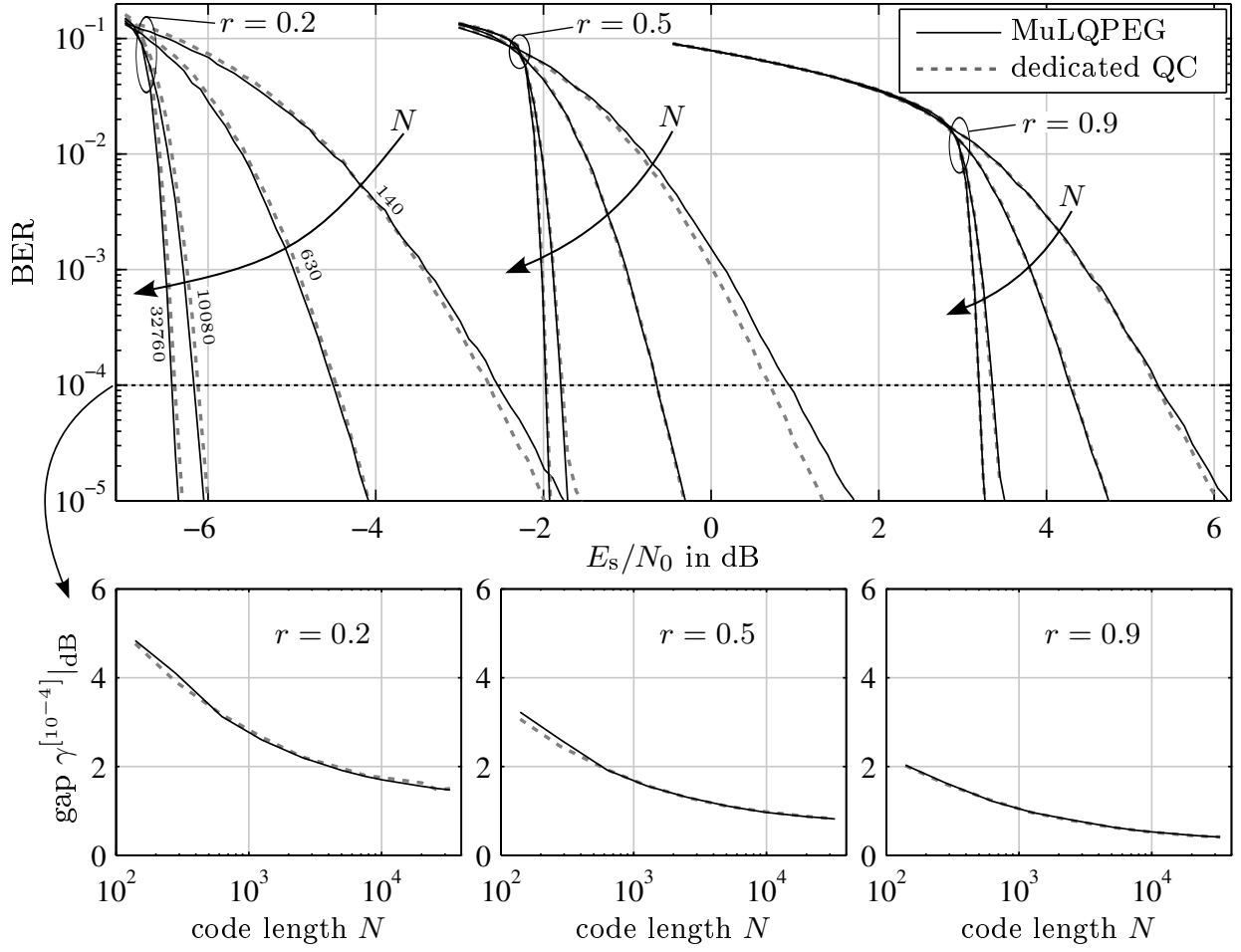


Figure 4.12: Top: BER over E_s/N_0 for $r \in \{0.2, 0.5, 0.9\}$. For both MuLQPEG and the dedicated case, curves are shown for $N \in \{140, 630, 10080, 32760\}$. Bottom: gaps $\gamma^{[10^{-4}]}_{\text{dB}}$ to capacity at BER 10^{-4} over code length N .

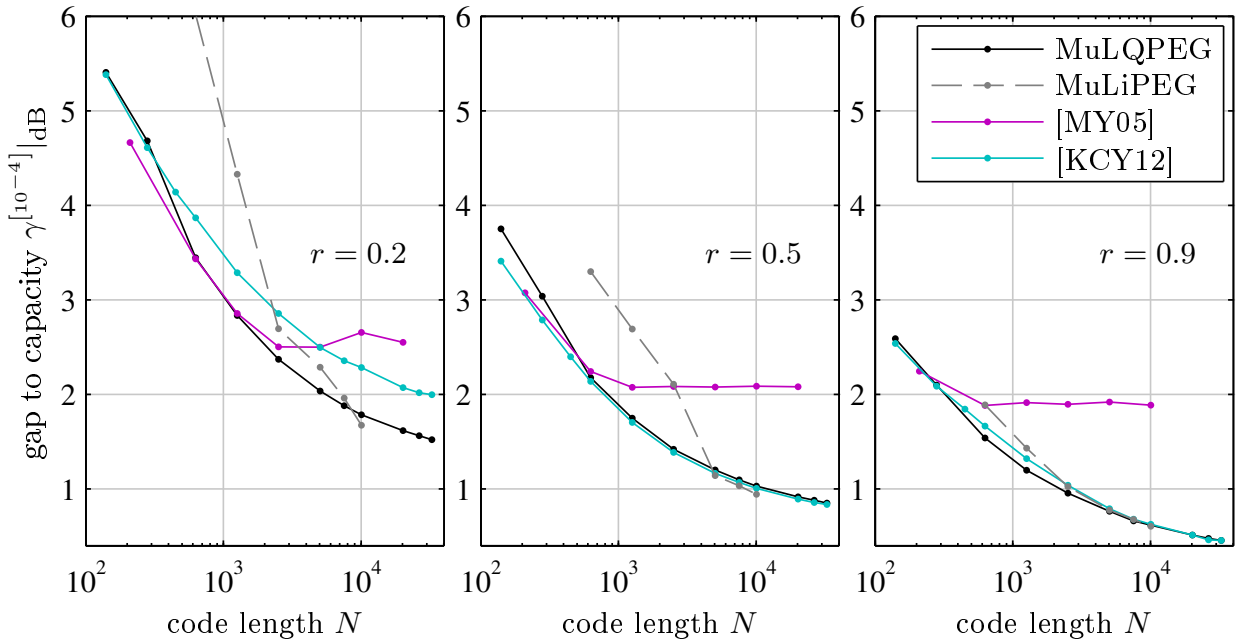


Figure 4.13: Gaps to capacity at BER 10^{-4} over code length N in logarithmic scale for MuLQPEG, MuLiPEG and two reference schemes.

similar performance to MuLQPEG for code rates $r = 0.5$ and $r = 0.9$. For $r = 0.2$, however a distinct performance loss is visible, which is mainly caused by the inferior degree distribution with above 80% variable nodes of degree two, that has to be used to construct an IRA code. Finally, for comparison the results of the MuLiPEG scheme from Section 4.1 are also included. Even though its results support a smaller range of code rates, the performance of MuLiPEG is distinctly inferior to that of the MuLQPEG scheme, especially at low code rates. As remedy, it is the only scheme that is generally applicable to LDPC codes and does not require a QC or IRA structure.

Besides the performance considerations that were just discussed, the reference schemes from the literature have a number of additional drawbacks. The method of [MY05] is not very flexible in terms of the supported code lengths, as each code length needs to be an integer multiple of the previous one. Furthermore, the successive construction of the lifting matrix has by far the highest complexity of all considered schemes⁸. The IRA scheme of [KCY12] is very flexible in terms of the code length but does not yield good performance for low code rates. Furthermore, a crucial assumption must be made for this method to yield a true multi-length code: if the concurrent shortening and puncturing is executed as proposed in [KCY12] a very large number of iterations (e.g., multiple thousand) needs to be conducted for the standard BP decoder to converge. Instead, a decoder that is capable of removing all punctured nodes from the Tanner graph and instead *merging* the two neighboring check nodes needs to be assumed. Whether or not such a decoder architecture exists and still fulfills the assumption of achieving multiple code lengths with only one encoder and decoder implementation is not further discussed here.

4.2.4 Multi-Rate and Multi-Length Quasi-Cyclic LDPC Codes⁹

The fact that the proposed optimization of a multi-length QC LDPC code is based on only a single mother lifting matrix description allows a straightforward integration of the multi-rate optimization based on shortening and puncturing presented in Section 3.3. If shortening and puncturing is considered solely based on the lifting matrix (i.e., shortening and puncturing groups of variable nodes instead of single nodes), it becomes independent of the actual lifting factor Z and is equally applicable to all code lengths supported by the multi-length QC LDPC code. The combination of these two methods, thus, results in a highly flexible multi-rate and multi-length QC LDPC code based on a single encoder and decoder implementation.

The asymptotic optimization of degree distributions and puncturing polynomials from Section 3.3.1 on pages 80 ff. can be directly applied to the case of QC LDPC codes by using the rate-compatible degree sequence from Algorithm 5 on page 90 during construction of the lifting matrix $\mathbf{A}$. Furthermore, all finite length considerations

⁸Executing the algorithm proposed in [MY05] to construct the code for code rate 0.9 and length 10080 took 13 hours. For the corresponding cases, the algorithm of [KCY12] runs a couple of minutes and the proposed MuLQPEG algorithm only takes a few seconds.

⁹The descriptions and results in this section follow the pre-published paper [BWV14].

(recoverability, check node concentration, ACE properties) that are considered in Section 3.3.2 on pages 89 ff. for the optimized check matrix construction MuRiPEG (Algorithm 6 on page 95) are equally valid for the check matrices $\mathbf{H}^{[Z]}$ that are expanded from the mother lifting matrix $\mathbf{A}$. By restricting all column and row selections during check matrix construction to a group-wise selection (which is automatically ensured by the circulant constraint, cf. Figure 4.9 on page 120), a QC LDPC check matrix can be constructed.

The proposed jointly optimized *multi-rate and multi-length quasi-cyclic improved PEG* (MuRMuLQiPEG) algorithm was pre-published in [BWV14] and combines the multi-rate scheme MuRiPEG (Algorithm 6 on page 95) with the multi-length quasi-cyclic approach MuLQPEG (Algorithm 7 on page 121). A full description is given in Algorithm 8.

The input comprises the lifting matrix dimensions $M' \times N'$, the supported lifting factors $Z_1 < \dots < Z_\xi$, the supported code rates, and the rate-compatible degree sequence $\mathbf{d}_v^{\text{RC}}$ that is determined by Algorithm 5 on page 90 together with the optimized shortened degree distributions and puncturing polynomials obtained by solving problem (3.51) on page 85. The processing order of variable node group degrees $\tilde{\mathbf{d}}_v^{\text{RC}} = (\tilde{d}_{v,1}^{\text{RC}}, \dots, \tilde{d}_{v,N'}^{\text{RC}})$ and the corresponding punctured positions $\tilde{\mathbf{p}} = (\tilde{p}_1, \dots, \tilde{p}_{N'})$ are determined according to the procedure in Algorithm 6 on page 95. Before the actual PEG-like processing starts, each punctured variable node group is connected to a unique check node group which later ensures recoverability of all variable nodes in that group (cf. Section 3.3.2 on page 89). The respective shift value is chosen randomly between 0 and the largest lifting factor Z_ξ as it does not influence recoverability. Then, the variable node groups are processed in reversed order, choosing the first connection of unpunctured groups randomly and the remaining $\tilde{d}_{v,n'}^{\text{RC}} - 1$ connections by successively extending a candidate set for all lifting factors in accordance with Algorithm 7 on page 121. Additionally, the ACE optimization from Section 3.3.2 on page 92 is included during the selection of candidates. For this, during tree expansion the value $\text{ACE}_m^{[Z_i]}(n)$ is determined for each reachable check node m , denoting the minimum ACE value of any shortest cycle passing through variable node n and check node m assuming an edge was placed between n and m . Punctured nodes are ensured not to be connected to indices larger than their first intentionally placed connection for guaranteed recoverability. The final lifting matrix needs to be permuted by $\mathbf{P}^{-1}$ to re-establish the original rate-compatible degree sequence $\mathbf{d}_v^{\text{RC}}$. Finally, the algorithm yields a single lifting matrix $\mathbf{A}$ that can be expanded by lifting factors $Z_1, \dots, Z_\xi$, and at the same time all resulting check matrices $\mathbf{H}^{[Z_i]}$ can be shortened from the left and punctured from the right (each in groups of Z_i bits) to realize the code rates $r_1, \dots, r_\zeta$, which is equivalently achieved by directly shortening and puncturing columns (i.e., variable node groups) of the lifting matrix.

Simulation Results

To confirm the enormous flexibility and the excellent error correction performance of the proposed MuRMuLQiPEG approach, a single mother lifting matrix with the following parameters is constructed:

Algorithm 8 Multi-Rate Multi-Length QC improved PEG (MuRMuLQiPEG)

```

1: Input:  $M', N'; Z_1, \dots, Z_\xi; r_1, \dots, r_\xi; \mathbf{d}_v^{\text{RC}} = (d_{v,1}, \dots, d_{v,N'})$  from Algorithm 5
2: With  $r_\mu = \frac{N'-M'}{N'}$ , mark the last  $\frac{r_\xi - r_\mu}{r_\xi} N'$  positions as punctured in vector  $\mathbf{p}$ 
3: Compute  $\tilde{\mathbf{d}}_v^{\text{RC}}$  by sorting the last  $\frac{1-r_\mu}{1-r_1} N'$  entries in  $\mathbf{d}_v^{\text{RC}}$  in ascending order
4: Let  $\mathbf{P}$  denote the permutation matrix such that  $\tilde{\mathbf{d}}_v^{\text{RC}} = \mathbf{d}_v^{\text{RC}} \mathbf{P}$ 
5: Compute the permuted punctured positions  $\tilde{\mathbf{p}} = \mathbf{p} \mathbf{P}$ 
6: Initialize  $\tilde{A}_{m',n'} = -1, \forall m', n'$  and check node group degrees  $d_{c,m'} = 0, \forall m'$ 
7: Initialize counter for punctured groups  $c = 0$ 
8: for  $n' = N', \dots, 1$  do
9:   if  $\tilde{\mathbf{p}}_{n'} = 1$  then
10:     Choose  $\tilde{A}_{n',M'-c}$  randomly from  $\{0, \dots, Z_\xi - 1\}$ 
11:     Set  $d_{c,M'-c} = 1$  and  $c = c + 1$ 
12:   end if
13: end for
14: Reset counter for punctured groups  $c = 0$ 
15: for  $n' = N', \dots, 1$  do
16:   if  $\tilde{\mathbf{p}}_{n'} = 0$  then
17:     Choose  $m' \in \arg \min_{\forall m'} d_{c,m'}$  and  $\tilde{A}_{m',n'} \in \{0, \dots, Z_\xi - 1\}$  randomly
18:     Update  $d_{c,m'} = d_{c,m'} + 1$ 
19:   end if
20:   for  $d = 2, \dots, \tilde{d}_{v,n'}^{\text{RC}}$  do
21:      $\mathcal{K}^{[Z_1]} = \bigcup_{\forall m': \tilde{A}_{m',n'} = -1} \{(m' - 1)Z_1 + 1, \dots, m'Z_1\}$ 
22:     for  $i = 1, \dots, \xi$  do
23:       Initialize  $\omega_m = \infty, \forall m$ 
24:       Using  $\mathbf{H}^{[Z_i]}$  and starting at variable node  $n = (n' - 1)Z_i + 1$ , expand a tree
       until no more nodes can be added, setting  $\omega_m = j$  for all check nodes  $m$  that
       are reached for the first time at depth  $j$ 
25:       if  $\tilde{\mathbf{p}}_{n'} = 1$  then Set  $\omega_m = 0, \forall m > (M' - c)Z_i$  to ensure recoverability
26:        $\mathcal{K}_{\text{girth}}^{[Z_i]} = \arg \max_{\forall m \in \mathcal{K}^{[Z_i]}} \omega_m; \mathcal{K}_{\text{weight}}^{[Z_i]} = \arg \min_{\forall m \in \mathcal{K}_{\text{girth}}^{[Z_i]}} d_{c,\lceil m/Z_i \rceil}$ 
27:        $\mathcal{K}_{\text{ACE}}^{[Z_i]} = \arg \max_{\forall m \in \mathcal{K}_{\text{weight}}^{[Z_i]}} \text{ACE}_m^{[Z_i]}(n)$ 
28:       if  $i < \xi$  then
29:          $\mathcal{K}^{[Z_{i+1}]} = \{m \in \{1, \dots, Z_{i+1}M'\} : \forall j \leq i, \exists \tilde{m} \in \mathcal{K}_{\text{ACE}}^{[Z_j]}, \lceil \frac{m}{Z_{i+1}} \rceil = \lceil \frac{\tilde{m}}{Z_j} \rceil$ 
30:          $\wedge s^{[Z_{i+1}]}(m) \bmod Z_j = s^{[Z_j]}(\tilde{m})\}$ 
31:       end if
32:       end for
33:       Choose  $m \in \mathcal{K}_{\text{ACE}}^{[Z_\xi]}$  randomly
34:        $\tilde{A}_{\lceil m/Z_\xi \rceil, n'} = s^{[Z_\xi]}(m)$  and  $d_{c,\lceil m/Z_\xi \rceil} = d_{c,\lceil m/Z_\xi \rceil} + 1$ 
35:     end for
36:   if  $\tilde{\mathbf{p}}_{n'} = 1$  then  $c = c + 1$ 
37: end for
38: The output is the permuted lifting matrix  $\mathbf{A} = \tilde{\mathbf{A}} \mathbf{P}^{-1}$ 

```

- $N' = 70$, $M' = 21$, code rate $r_\mu = 0.7$,
- lifting factors $Z \in \{3, 9, 18, 36, 72, 144, 288\}$ resulting in expanded code lengths $N \in \{210, 630, 1260, 2520, 5040, 10080, 20160\}$,
- shortened code rates $r \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ and punctured code rates $r \in \{0.8, 0.9\}$,
- degree distributions and puncturing polynomials from optimized multi-rate ensemble with maximum variable node degree $d_v = 10$ from Section 3.3.1.

For comparison, for each of the 63 combinations of code lengths and rates, a dedicated PEG code of the same resulting effective code length and rate and with an optimized degree distribution of same maximum degree is constructed. BER simulations are conducted with BP decoding and a maximum of $I_{\max} = 100$ decoding iterations and gaps of the BER curves to capacity are measured at BER 10^{-4} .

In Figure 4.14 one plot per mother code length N is shown, containing BER over E_s/N_0 curves for all code rates r for both MuRMuLQiPEG and the dedicated PEG codes. Additionally, in the bottom right corner one curve per code length of the gaps to capacity $\gamma^{[10^{-4}]}_{\text{dB}}$ at BER 10^{-4} over the code rate is depicted. The difference of the mean gaps $\Delta\bar{\gamma}^{[10^{-4}]}_{\text{dB}} = \bar{\gamma}^{[10^{-4}]}_{\text{MuRMuLQiPEG}}_{\text{dB}} - \bar{\gamma}^{[10^{-4}]}_{\text{ded}}_{\text{dB}}$ of both approaches is also given for each code length. The results illustrate, that the proposed scheme performs very close to the case of dedicated optimized PEG codes over the range of code rates from 0.1 to 0.9 and code lengths from 210 to 20160. Averaging over all code rates and lengths, the total penalty is only 0.076 dB, which effectively means that it is possible to replace 63 dedicated codes by only a single multi-rate and multi-length code with almost no performance loss.

4.2.5 Concluding Remarks: MuLQPEG and MuRMuLQiPEG

The multi-length QC LDPC codes (MuLQPEG) presented in this section support multiple code lengths by using a fixed mother lifting matrix that can be expanded by multiple lifting factors. The only prerequisite for these multi-rate codes is an encoder and decoder architecture that is based only on the lifting matrix instead of the fully expanded check matrix. A variety of such architectures have been presented and well investigated in the literature, as nearly all LDPC codes used in practical systems are QC LDPC codes.

The multi-length QC LDPC codes constructed by MuLQPEG are capable of achieving very wide ranges of code lengths (e.g., 140 to 32760) with only negligible performance loss compared to the case of using one dedicated LDPC code for each code length. Furthermore, a straightforward combination of these multi-length codes with the multi-rate features presented in Chapter 3 leads to a flexibly adaptable multi-rate and multi-length QC LDPC code (MuRMuLQiPEG). In an exemplary scenario of supporting code rates from 0.1 to 0.9 and code lengths from 210 to 20160, a single mother code designed by the proposed MuRMuLQiPEG method performs (on average) within 0.076 dB compared to a set of 63 dedicated optimized codes.

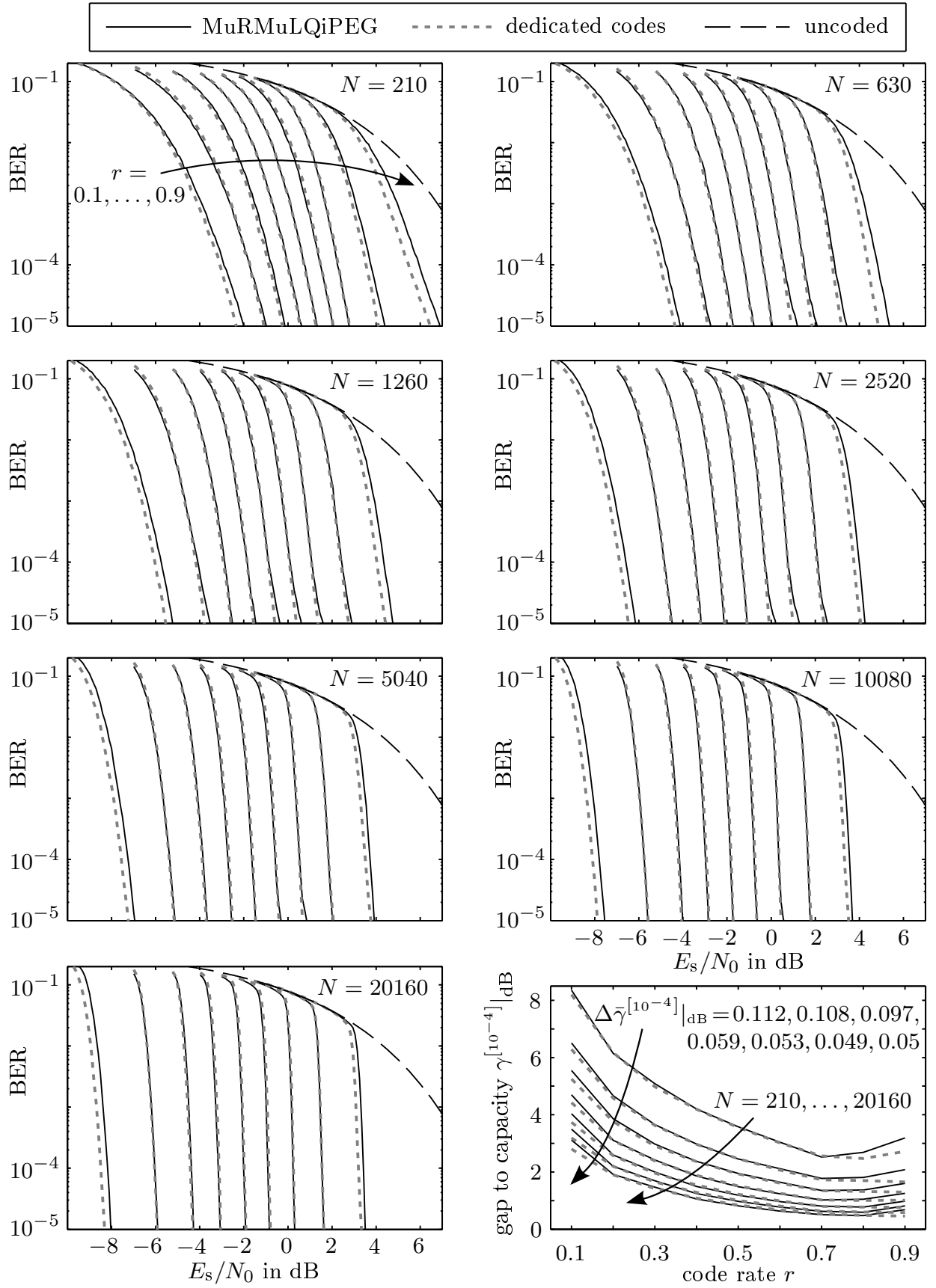


Figure 4.14: Each plot shows BER curves for all code rates. The bottom right plot shows BER gaps to capacity over code rate and mean gaps $\Delta \bar{\gamma}_{[10^{-4}]}|_{\text{dB}}$ of MuRMuLQiPEG to the dedicated case for all code lengths.

Furthermore, the proposed method MuLQPEG, that is based on QC LDPC codes, results in better performance than the MuLiPEG method from Section 4.1, that realizes length reduction by concurrent shortening and puncturing. The advantage of the latter one, however, is that it does not rely on a specific code structure and is, thus, applicable to arbitrary LDPC codes.

Altogether, the proposed MuLQPEG and MuRMuLQiPEG methods enable new degrees of flexibility in the design of FEC schemes for communication systems and at the same time significantly reduces hardware requirements.

Optimized Message-Passing Schedules

The standard BP decoder uses the *flooding* schedule as introduced in Section 2.5: first the information for all the check nodes is evaluated and then “flooded” to the variable nodes. Then all the variable nodes are evaluated, and the resulting information is flooded back to the check nodes. Several different scheduling approaches for improving the BP performance have been presented in the literature. A *probabilistic schedule* has been introduced in [MB01a, MB01b]. This approach takes into account the local girth of each node. To each variable node a probability depending on the local girth is assigned and at each iteration, the outgoing messages of each variable node are updated with this probability, whereas outgoing check node messages are always updated. This approach improves the decoding performance of LDPC codes in terms of residual BER, however, the convergence properties slightly change for the worse.

In order to improve the convergence properties of BP decoding, manifold *sequential schedules* have been presented in the literature, e.g. [KK03, ZF02, MS03, Hoc04, SLG04, SLG07]. Two examples are *shuffled* BP decoding [ZF02] and the dual *check-shuffled* or *layered* BP decoding [SLG04, Hoc04]. These algorithms exploit the fact that new information obtained while updating single nodes can immediately be used in the updates of other nodes within the same iteration. This can lead to a faster decoding convergence and thus to a better BER performance in the case of a constrained number of iterations.

More recently, in [CGW07a, CGW07b, CGW10] *Informed Dynamic Scheduling* (IDS) was introduced as the first message passing schedule for LDPC codes that dynamically updates those parts of the graph that have not converged yet. IDS determines the order of messages to be updated and propagated using a metric called *residual*, measuring how far a message is from convergence. It was shown that this dynamic approach has the capability of converging with much fewer message updates than other schedules and, in addition, is able to solve many so-called *trapping set errors* [Ric03], enabling them to achieve lower error rates than traditional techniques even if a large number of iterations is carried out.

In this chapter, a number of novel decoder schedules, which have been pre-published in [BSV11, BV11], are presented and evaluated. First, two *probabilistic sequential schedules* combining fast convergence with improved error correction performance

are introduced in Section 5.1. Additionally, a new class of *dynamic schedules* is established in Section 5.2, which greatly reduces the complexity of previously proposed dynamic schedules and at the same time maintains and in some cases even surpasses their error correction capabilities.

5.1 Probabilistic Sequential Decoding¹

It has been shown in [SLG07] that by sequentially updating and propagating messages in a node-by-node manner throughout the decoding process, the convergence speed of the BP decoder can be up to twice as fast as for the flooding strategy. This is explained by the faster spreading of information when reusing extrinsic information that is generated during the update of single nodes already within the same decoding iteration. In [YEK⁺07] and [VDV⁺10] the effects of this sequential updating strategy has also been examined for the decoding of non-binary LDPC codes over $\mathbb{F}_q$. Similarly to the binary case, the decoding process can be accelerated by a factor of up to two.

As pointed out in [MB01a, MB01b], it is advantageous to consider the structure of the Tanner graph of a code in the decoding process. As mentioned in Chapter 2, it is well-known that the presence of short cycles degrades the performance of BP decoding. Let $g_{\max} = \max_{\forall n} g_n$ denote the maximal local girth g_n occurring at any variable node n . After $g_n/2$ iterations the information in a cycle of length g_n is propagated back to its origin, and the assumption of statistical independence of the incoming extrinsic knowledge and the own knowledge does no longer hold. By only updating variable node n with probability $P_n^{[g]} = g_n/g_{\max}$ in each iteration, the violation of the statistical independence can on average be postponed.

In the pre-published paper [BSV11], a combination of the shuffled [ZF02] and check-shuffled [SLG04] approaches with the girth-based probabilistic approach [MB01a, MB01b] has been introduced for the decoding of both binary and non-binary LDPC codes and is described in the following. It is shown that the performance in terms of BER can be improved compared to other sequential scheduling strategies with (almost) no additional operations. The drawback is a slight loss in convergence speed, however, the convergence is still faster than for the standard BP decoding with simple flooding and the standard probabilistic parallel approach of [MB01b].

5.1.1 Probabilistic Shuffled Scheduling

The first proposed scheduling strategy follows the idea of the shuffled schedule [ZF02] and adds a probabilistic component similar to the one presented in [MB01b]. In a pre-processing step the local girth g_n of each variable node n is calculated and divided by the maximal local girth $g_{\max}$ to yield the updating probability $P_n^{[g]} = g_n/g_{\max}$. All

¹The description of probabilistic sequential decoding in this section follows the one given in [BSV11].

variable-to-check messages $v_{n,m}^{(0)}$ are initially set to the channel LLRs L_n and all check-to-variable messages $c_{m,n}^{(0)}$ are set to zero. At the beginning of the i -th iteration all check-to-variable messages $c_{m,n}^{(i-1)}$ and all variable-to-check messages $v_{n,m}^{(i-1)}$ are available as result of the previous iteration $i-1$. In each iteration the decoder traverses all variable nodes in a serial fashion and randomly decides with probability $P_n^{[g]}$ whether to update the incoming and outgoing messages of each node or not.² The probabilistic shuffled scheduling is described in Algorithm 9.

It should be noted that the iteration counter i does not exactly correspond to the number of iterations in a flooding or a simple serial schedule, since – depending on the girth structure of the code – this schedule performs fewer message updates per iteration as some variable nodes will not be updated due to the probabilistic component. To allow for a fair comparison of the different schedules, the actual number of message updates is counted and denoted U , and the number of equivalent (flooding) iterations needed by a schedule is defined as $I_{\text{eff}} = \lfloor \frac{U}{2E} + \frac{1}{2} \rfloor$, with E denoting the number of edges in the Tanner graph and $\lfloor x \rfloor$ denoting the largest integer smaller than or equal to x .

Algorithm 9 Probabilistic Shuffled Belief Propagation

- 1: Initialize $v_{n,m}^{(0)} = L_n, \forall n, m$
 - 2: Initialize $c_{m,n}^{(0)} = 0, \forall n, m$
 - 3: Initialize $i = 1$
 - 4: **while** stopping rule not satisfied **do**
 - 5: **for** $n = 1, \dots, N$ **do**
 - 6: **if** $\text{rand}(0, 1) \leq P_n^{[g]}$ **then**
 - 7: Compute all incoming messages of variable node n , i.e. $\forall m \in \mathcal{M}(n)$:

$$c_{m,n}^{(i)} = 2 \operatorname{atanh} \left(\prod_{\substack{n' \in \mathcal{N}(m) \\ n' < n}} \tanh \left(\frac{v_{n',m}^{(i)}}{2} \right) \cdot \prod_{\substack{n' \in \mathcal{N}(m) \\ n' > n}} \tanh \left(\frac{v_{n',m}^{(i-1)}}{2} \right) \right)$$
 - 8: Compute all outgoing messages of variable node n , i.e. $\forall m \in \mathcal{M}(n)$ process $v_{n,m}^{(i)}$ according to the variable node update equation (2.42)
 - 9: **else**
 - 10: Set $c_{m,n}^{(i)} = c_{m,n}^{(i-1)}$ and $v_{n,m}^{(i)} = v_{n,m}^{(i-1)}$ for all $m \in \mathcal{M}(n)$
 - 11: **end if**
 - 12: **end for**
 - 13: Start next iteration by setting $i = i + 1$
 - 14: **end while**
 - 15: Output hard decision $\hat{y}_n, \forall n$ according to (2.43)
-

²The random decision can be done using an arbitrary (pseudo-) random number generator or even by reading pseudo-random numbers one after another from a precomputed look-up table (and restarting at its beginning once the end has been reached). The only requirement regarding the size of the look-up table is that it should be coprime with the number of variable nodes, to avoid the same random number to be used for the same variable node in each iteration.

The formal description of this schedule is similar to the *lazy scheduling* approach presented in [LSL07]. However, in *lazy scheduling* an updating probability is chosen dynamically according to some reliability measure to avoid unnecessary message computations and reduce the overall complexity of the decoding process. The presented approach of probabilistic shuffled scheduling, by contrast, uses fixed (girth dependent) updating probabilities and accepts a slight increase in the overall decoding complexity while it aims at achieving lower residual bit error rates.

5.1.2 Probabilistic Check-Shuffled Scheduling

The second proposed scheduling is described as a dual version of the probabilistic shuffled scheduling from the previous section, iterating over the check nodes rather than the variable nodes, just as the scheduling in [SLG04] is described as a dual version of the shuffled schedule in [ZF02]. Again the updating probability of variable node n is denoted $P_n^{[g]}$. Further, let $\mathcal{V}_{\text{active}}^{(i)}$ denote the set of variable nodes that participate in updates of iteration i , i.e., the *active* variable nodes. After the same initialization as for the probabilistic shuffled schedule, at the beginning of each iteration the decoder decides for each variable node whether it is active or not (i.e., whether it sends and receives updated messages) with probability $P_n^{[g]}$. Then all check nodes are traversed serially, and incoming and outgoing messages are only updated for those edges that are connected to active variable nodes. This probabilistic check-shuffled scheduling is described in Algorithm 10.

Again the total number of message updates U is counted and the number of equivalent (flooding) iterations I_{eff} is determined as described for the probabilistic shuffled schedule to allow a fair comparison to other schedules.

5.1.3 Adaptation to Decoding of Non-Binary LDPC Codes

The difference between the Tanner-graph representation of binary LDPC codes and non-binary LDPC codes over $\mathbb{F}_q$ is that in the non-binary case a weight according to the value of the corresponding check matrix element is assigned to each edge of the (weighted) Tanner-graph. Still, the definition of the local girth as the length of the shortest cycle passing through a node is the same in both cases, since it is independent of the edge weights. Furthermore, the presence of (short) cycles causes the same degradation on the decoding performance due to the statistical dependence of messages after a certain number of decoding iterations. With this analogy it can be expected that the additional probabilistic component, taking into account the girth structure of a code has the same effect on the decoding of both binary and non-binary LDPC codes. Results of [YEK⁺07] and [VDV⁺10] indicate that applying serial updating schemes to the decoding of non-binary LDPC codes has the same advantage of increasing the convergence speed as for the binary case.

In analogy to [BSV11], the effects of the probabilistic schedule of [MB01b] and the novel probabilistic shuffled schedule (Section 5.1.1) are investigated for an adaptation

Algorithm 10 Probabilistic Check-Shuffled Belief Propagation

```

1: Initialize  $v_{n,m}^{(0)} = L_n$  and  $c_{m,n}^{(0)} = 0$ ,  $\forall n, m$ 
2: Initialize  $i = 1$ 
3: while stopping rule not satisfied do
4:   Set  $\mathcal{V}_{\text{active}}^{(i)} = \emptyset$ 
5:   for  $n = 1, \dots, N$  do
6:     if  $\text{rand}(0, 1) \leq P_n^{[g]}$  then
7:       Activate variable node  $n$ , i.e., set  $\mathcal{V}_{\text{active}}^{(i)} = \mathcal{V}_{\text{active}}^{(i)} \cup \{n\}$ 
8:     end if
9:   end for
10:  for  $m = 1, \dots, M$  do
11:    for  $n \in \mathcal{N}(m)$  do
12:      if  $n \in \mathcal{V}_{\text{active}}^{(i)}$  then
13:        Compute  $v_{n,m}^{(i)} = L_n + \sum_{\substack{m' \in \mathcal{M}(n) \\ m' < m}} c_{m',n}^{(i)} + \sum_{\substack{m' \in \mathcal{M}(n) \\ m' > m}} c_{m',n}^{(i-1)}$ 
14:      else
15:        Set  $v_{n,m}^{(i)} = v_{n,m}^{(i-1)}$ 
16:      end if
17:    end for
18:    for  $n \in \mathcal{N}(m)$  do
19:      if  $n \in \mathcal{V}_{\text{active}}^{(i)}$  then
20:        Compute  $c_{m,n}^{(i)}$  according to the check node update equation (2.41)
21:      else
22:        Set  $c_{m,n}^{(i)} = c_{m,n}^{(i-1)}$ 
23:      end if
24:    end for
25:  end for
26:  Start next iteration by setting  $i = i + 1$ 
27: end while
28: Output hard decision  $\hat{y}_n$ ,  $\forall n$  according to (2.43)

```

to the non-binary case. For this purpose, the non-binary BP decoding algorithm as presented in Section 2.10.2 is modified to allow for the employment of different schedules. Results for the combination of the non-binary BP decoder with flooding, probabilistic, shuffled, and probabilistic shuffled schedules are presented below.

5.1.4 Simulation Results

Binary LDPC Codes

In order to evaluate the performance of both probabilistic sequential (i.e., shuffled and check-shuffled) approaches for the binary case, the WiMAX LDPC code of rate $r = 1/2$ and length $N = 576$ is considered.

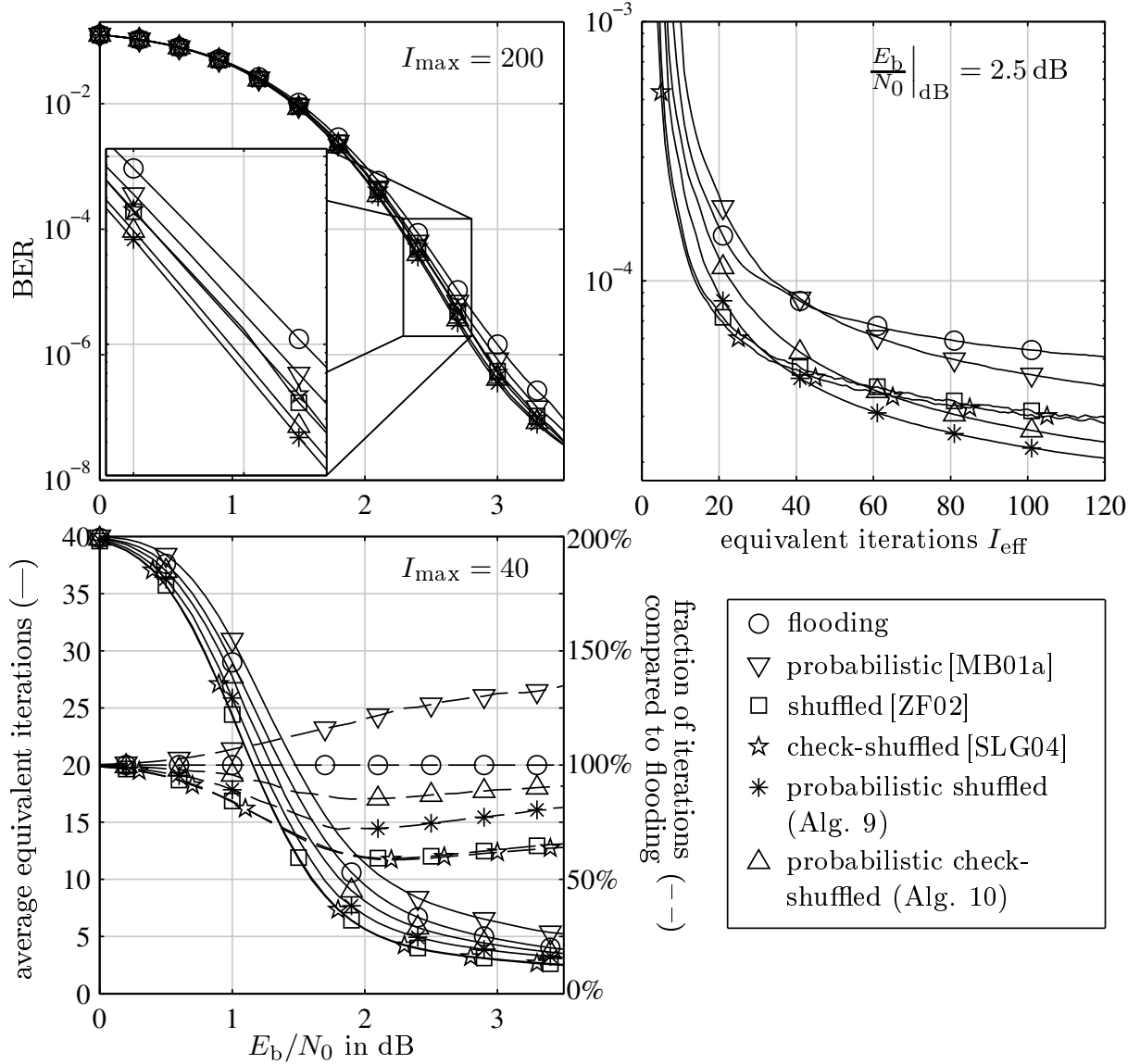


Figure 5.1: Different schedules for BP decoding of WiMAX rate-1/2 length 576 LDPC code. Top left: BER performance at $I_{\max} = 200$ iterations. Top right: Convergence at 2.5 dB. Bottom left: Average number of equivalent iterations (solid) and fraction of iterations compared to flooding schedule (dashed) at $I_{\max} = 40$ iterations.

The top left plot in Figure 5.1 depicts the BER as a function of E_b/N_0 . It can be seen that the probabilistic shuffled and probabilistic check-shuffled algorithms outperform the other four schedules (flooding, probabilistic, shuffled, and check-shuffled) if a large number of iterations (here $I_{\max} = 200$) is executed. In the top right plot of Figure 5.1 the convergence (BER as a function of the number of equivalent iterations) at $E_b/N_0|_{\text{dB}} = 2.5$ dB is shown. As expected, the convergence of the probabilistic (check-) shuffled schedules is slightly slower than for the non-probabilistic (check-) shuffled approaches [ZF02, SLG04], however it still outperforms the flooding and probabilistic [MB01a] schedules. For more than 30 iterations, the probabilistic shuffled schedule also slightly outperforms the shuffled schedule and yields the lowest BER. The bottom left plot in Figure 5.1 shows the average number of equivalent decoding

iterations needed for convergence against the E_b/N_0 for the different schedules (solid lines). Here, the maximum number of iterations is only $I_{\max} = 40$. The dashed lines show the ratio of the number of iterations needed by the flooding schedule and the number of iterations needed by the other schedules. It can be seen that the probabilistic (check-) shuffled schedules require slightly more (equivalent) iterations than the non-probabilistic (check-) shuffled schedules, but the saving compared to the flooding schedule is still as high as 28 %.

Although the probabilistic check-shuffled schedule performs slightly worse than the probabilistic shuffled schedule in this example, it exhibits the same advantages regarding implementation aspects as indicated for the non-probabilistic check-shuffled approach in [SLG04]. Finally, it should be noted that strategies like *group shuffling* [ZF02] (allowing parallelization) can be directly integrated into the probabilistic (check-) shuffled schedules.

Non-Binary LDPC Codes

For the evaluation of the non-binary case, two non-binary PEG codes from [HEA05] are considered. The non-binary BP decoding algorithm from Section 2.10.2 is used with different schedules. Results of flooding, probabilistic [MB01b], shuffled [ZF02], and the probabilistic shuffled (Section 5.1.1) schedules are shown for two codes over $\mathbb{F}_8$ and $\mathbb{F}_{64}$ with rate 1/2 and a binary code length of $N = 1008$ information bits. Results for the check-shuffled and the proposed probabilistic check-shuffled schedules are not shown here since they are similar to those of the shuffled and probabilistic shuffled approaches.

The top left plot in Figure 5.2 shows the BER as a function of E_b/N_0 with a maximum number of $I_{\max} = 100$ decoding iterations. Similarly to the binary case the probabilistic shuffled schedule outperforms the three other schedules for both codes and for all shown channel qualities. In the top right plot of Figure 5.2, the convergence at $E_b/N_0|_{\text{dB}} = 1.9$ dB is depicted. Again a behavior similar to the binary case can be observed. The initial convergence speed is slightly higher for the non-probabilistic schedules (flooding and shuffled), but after a certain number of iterations, lower bit error rates can be achieved by the probabilistic schedules. For more than 35 (resp. 40) iterations the probabilistic shuffled schedule yields the lowest bit error rate for the code over $\mathbb{F}_8$ (resp. $\mathbb{F}_{64}$). Finally, in the bottom left corner of Figure 5.2 the average number of equivalent iterations needed by the non-binary BP decoder with different schedules is shown for varying channel qualities E_b/N_0 . For the sake of clarity, the results of the code over $\mathbb{F}_{64}$ are omitted in this figure. However, they show the same characteristics as those for the $\mathbb{F}_8$ code with the difference that the absolute number of iterations needed is slightly lower for the $\mathbb{F}_{64}$ code. This is mainly due to the fact that for larger Galois fields a smaller non-binary code length N_q is required to achieve the same binary code length N . The probabilistic shuffled schedule needs slightly more (equivalent) iterations than the shuffled schedule but still up to 28 % can be saved compared to the flooding schedule.

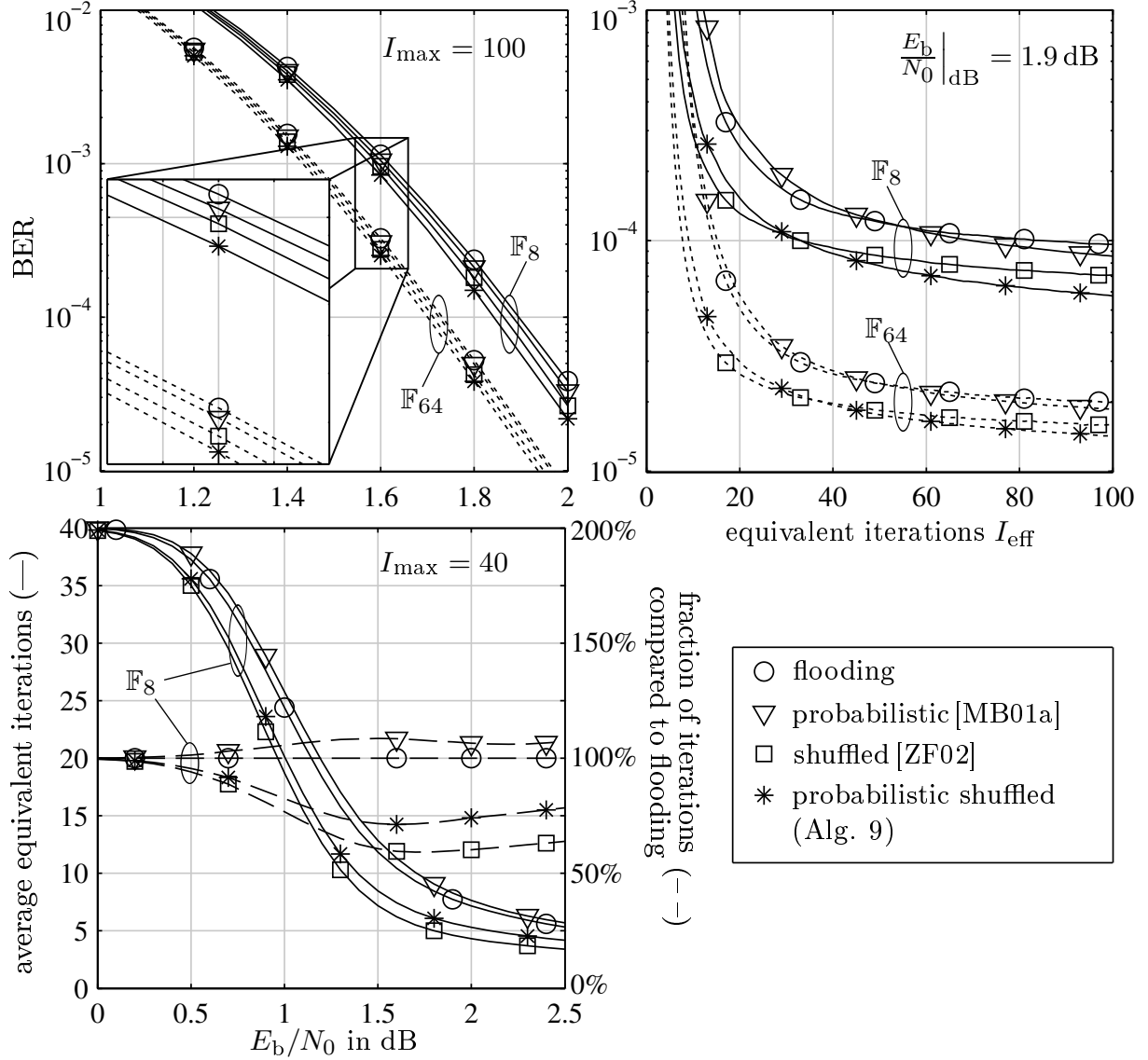


Figure 5.2: Different schedules for non-binary BP decoding of two non-binary rate-1/2 LDPC codes over $\mathbb{F}_8$ and $\mathbb{F}_{64}$ with binary code length $N = 1008$. Top left: BER performance at $I_{\max} = 100$ iterations. Top right: Convergence at 1.9 dB. Bottom left: Average number of equivalent iterations (solid) and fraction of iterations compared to flooding schedule (dashed) at $I_{\max} = 40$ iterations for the code over $\mathbb{F}_8$ only.

5.2 Improved Residual Belief Propagation³

The first instance of a so-called *informed dynamic scheduling* (IDS) strategy for general message-passing algorithms – and not especially related to LDPC codes – was presented in [EMK06] under the name of *Residual Belief Propagation* (RBP). This approach was adopted and extended for the decoding of LDPC codes in [CGW07a, CGW07b, CGW10]. IDS approaches are based on the so called *residual* which is defined as the norm of the difference between a message before and after an update.

³The descriptions in this section follow the pre-published results in [BV11].

The residual $\mathfrak{r}(c_{m,n})$ of a check-to-variable message $c_{m,n}$ is defined as [CGW10]

$$\mathfrak{r}(c_{m,n}) = \|c_{m,n}^{\text{updated}} - c_{m,n}^{\text{current}}\|. \quad (5.1)$$

The current value of the message is denoted by $c_{m,n}^{\text{current}}$ and the value if it was updated at this stage is denoted by $c_{m,n}^{\text{updated}}$. In case of binary codes, the messages are scalar LLRs and the norm corresponds to the absolute value. From this definition it directly follows that messages with large residuals are still far from convergence, while small residuals indicate messages that have already converged to a final state. In IDS the residuals are used to determine the order in which messages are updated. By only updating messages with large residuals, convergence of the decoding process has been observed to be achieved with much fewer overall message computations than for traditional (flooding or sequential) message-passing schedules. It was also shown that certain IDS strategies can correct more errors than other schedules since they are able to overcome certain *trapping sets* [Ric03].

The shortcoming of IDS, however, is that the computation of residuals and the selection of the messages to be updated cause a significant increase in computational complexity. Many more residuals are calculated than messages are updated, and for each message update all residuals have to be searched through beforehand. The first problem has been addressed in [CGW07a, CGW07b, CGW10]. Each calculation of a residual requires the evaluation of the BP update equations (2.41) or (2.42). However, the computed values are only needed to determine the order of message updates. To reduce this additional complexity it was proposed to use the min-sum approximation [FMI99] for the calculation of residuals, while still using the exact BP equations for the actual message update. This approach was shown to reduce the complexity without losing error correction or convergence speed performance as compared to using exact calculation of residuals. While this reduction of overhead caused by the computation of residuals has been addressed already, the pre-published paper [BV11] was the first work in which efforts are made to reduce the second component of additional complexity, which is the continuous search for large residuals.

Two novel decoding algorithms from [BV11] are presented in the following, which address the issue of computational complexity by significantly reducing the number of search operations needed per equivalent iteration. Furthermore, it is shown that these algorithms maintain – and in some cases even exceed – the convergence speed and also the error rate performance of previously known IDS strategies.

At first, the schemes RBP and Node-Wise RBP (NWRBP) from [CGW07a] are described. Following this, the two novel dynamic schedules *lazy queue residual decoding* (LQRD) and *queue residual decoding* (QRD) are presented.

5.2.1 Residual Belief Propagation (RBP)

RBP was introduced in [CGW07a] and is a greedy message-passing schedule, which always updates the check-to-variable message with the largest residual as described in Algorithm 11. In accordance with [CGW07a], E executions of the main loop (lines

Algorithm 11 Residual Belief Propagation

```

1: Initialize all  $v_{n,m} = L_n$ 
2: Initialize all  $c_{m,n} = 0$ 
3: Compute all  $\mathbf{r}(c_{m,n})$  according to (5.1)
4: while stopping rule not satisfied do
5:   Find message  $c_{m,n}$  with largest residual  $\mathbf{r}(c_{m,n})$ 
6:   Update message  $c_{m,n}$ 
7:   Set  $\mathbf{r}(c_{m,n}) = 0$ 
8:   for  $m' \in \mathcal{M}(n) \setminus m$  do
9:     Update message  $v_{n,m'}$ 
10:    for  $n' \in \mathcal{N}(m') \setminus n$  do
11:      Compute  $\mathbf{r}(c_{m',n'})$ 
12:    end for
13:  end for
14: end while

```

4 – 14) are defined as one equivalent iteration, where E denotes the number of edges in the Tanner graph. With this definition, in one equivalent iteration of RBP the same number of check-to-variable messages is updated as in one iteration of a conventional (flooding or serial) schedule. In the following, this criterion of equal numbers of check-to-variable message updates is used to define an equivalent iteration for all considered schedules. At first glance this definition does not account for the fact that in RBP after a single check-to-variable message $c_{m,n}$ is updated (line 6), $d_{v,n} - 1$ variable messages have to be updated (line 9). However, these variable message updates can be realized by a single addition each, since only one incoming message of each variable node n has changed, so that the outgoing messages to all check nodes except m will vary by the same value as the incoming message $c_{m,n}$. Therefore, the complexity of those $d_{v,n} - 1$ variable message updates is the same as for the computation of one conventional variable message update in which $d_{v,n}$ incoming values are added up.

It can be seen that the additional complexity introduced by RBP is caused by the search for the largest residual (line 5) and the calculation of residuals (lines 10 – 12). Since each calculation of a residual requires the evaluation of the check node update equation (2.41), but is only needed to determine the order of message updates, *Approximate RBP* (ARBP) was introduced in [CGW07a] to reduce the computational overhead. In ARBP the min-sum approximation [FMI99] is used instead of (2.41) for the calculation of residuals, while the actual message updates still use the exact BP equations (2.41) and (2.42). This approach was shown to perform nearly as well as the one using exact residuals.

However, it turns out that for medium to large code lengths (e.g., larger than 1000) the complexity of the search for the largest residual dominates the whole RBP decoding, since it raises the overall decoding complexity from $\mathcal{O}(N)$ to $\mathcal{O}(N^2)$. This is due to the fact that for each message update, a list of length E has to be searched. In fact, if the residuals are organized as an ordered list and reordered whenever a residual changes, it is possible to implement RBP with complexity $\mathcal{O}(N \cdot \log(N))$. Nevertheless, the

linear complexity property is lost and with increasing N a significant amount of the computational overhead is caused by this managing of an ordered list.

Node-Wise RBP (NWRBP)

Another drawback of RBP originates from its greediness. As shown in [CGW07b], certain types of errors cannot easily be corrected by RBP causing a distinct error floor behavior. A partial remedy to this behavior is accomplished by what was termed *node-wise RBP* (NWRBP) in [CGW07a] and simply *node-wise scheduling* (NS) later in [CGW10]. In this less greedy approach, not only the message having the largest residual is updated in each step, but all outgoing messages of the corresponding check node instead. A more detailed description of NWRBP is not included here but can be found in [CGW07a] and [CGW10]. It was shown that NWRBP can reach lower error rates than other strategies while converging only slightly slower than RBP.

Nevertheless, the computational overhead of calculating residuals and searching (or sorting) a list is similar to RBP. While the same number of residuals has to be calculated per iteration, the number of search operations is slightly reduced since for each largest residual that was found on average $\bar{d}_c$ check messages instead of just one are updated. The average check node degree $\bar{d}_c$, however, is only a small constant compared to the number E of edges in the Tanner graph, such that the overall complexity is only slightly reduced.

5.2.2 Novel Queue Residual Decoding

Two novel flexible dynamic schedules, pre-published in [BV11], that are capable of significantly reducing the average number of search operations executed per iteration are presented here. At the same, time both of these approaches can overcome the problem of RBP's greediness to an even greater extent than NWRBP does. Furthermore, while NWRBP converges slower than RBP, the presented approaches converge as fast as RBP and show superior performance even when the maximum number of decoding iterations is very small.

The proposed schedules are based on the idea of not only updating one single message that has the largest residual $\mathbf{r}_{\max}$, but updating all messages whose residuals are greater than an adaptive threshold $\mathbf{r}_{\text{thresh}} = \alpha \cdot \mathbf{r}_{\max}$ which is determined by a factor α (with $0 \leq \alpha \leq 1$) and the current value $\mathbf{r}_{\max}$. For this purpose an updating queue $\mathcal{Q}$ is introduced, to which only messages with residuals greater than the current threshold are appended. In the following the two novel approaches are described in detail.

Lazy Queue Residual Decoding (LQRD)

LQRD is described in Algorithm 12 and is based on RBP as it likewise updates a single check message in each execution of the main loop (lines 4 – 18). In contrast to RBP, however, this message is not selected by searching through all residuals but simply obtained from the head of a message queue $\mathcal{Q}$. At the start of decoding, $\mathcal{Q}$

Algorithm 12 Lazy Queue Residual Decoding (LQRD)

```

1: Initialize all  $v_{n,m} = L_n$ 
2: Initialize all  $c_{m,n} = 0$ 
3: Compute all  $\mathbf{r}(c_{m,n})$ 
4: while stopping rule not satisfied do
5:   if  $\mathcal{Q}$  is empty then
6:     Find message with largest residual  $\mathbf{r}_{\max} = \max_{\forall m,n} \mathbf{r}(c_{m,n})$  and append it to  $\mathcal{Q}$ 
7:     Append all messages  $\mathbf{r}(c_{m,n})$  with  $\mathbf{r}(c_{m,n}) > \alpha \cdot \mathbf{r}_{\max}$  to  $\mathcal{Q}$ 
8:   end if
9:   Remove message  $c_{m,n}$  from head of  $\mathcal{Q}$ 
10:  Update message  $c_{m,n}$ 
11:  Set  $\mathbf{r}(c_{m,n}) = 0$ 
12:  for  $m' \in \mathcal{M}(n) \setminus m$  do
13:    Update message  $v_{n,m'}$ 
14:    for  $n' \in \mathcal{N}(m') \setminus n$  do
15:      Compute  $\mathbf{r}(c_{m',n'})$ 
16:    end for
17:  end for
18: end while

```

is initialized and only modified when it runs empty. Every time this happens, the value $\mathbf{r}_{\max}$ of the currently largest residual is determined and all messages with residuals $\mathbf{r}(c_{m,n}) > \alpha \cdot \mathbf{r}_{\max}$ are appended to $\mathcal{Q}$. Then, picking messages one after another from the head of $\mathcal{Q}$, message updates are performed until either the stopping rule is satisfied or the queue runs empty again. This procedure is motivated as follows.

If at some stage of decoding there are large outliers in the values of the residuals, LQRD will focus on updating the corresponding messages first since the threshold $\mathbf{r}_{\text{thresh}} = \alpha \cdot \mathbf{r}_{\max}$ for a message to be added to $\mathcal{Q}$ will be very high and consequently only the outliers will be updated. This conforms with the original idea of RBP which enables convergence with very few overall message updates. If, on the other hand, there are no outliers, RBP will spend a lot of effort on finding the largest residual which will, in fact, not be much different from, e.g., the second or third largest residuals. In these cases, LQRD will behave differently, since the threshold $\mathbf{r}_{\text{thresh}}$ will be relatively small and many messages will be added to $\mathcal{Q}$. Thus, while still constantly focusing on parts of the graph that have not yet converged, LQRD is far less greedy (depending, of course, on the choice of α) than RBP or even NWRBP.

It can easily be seen that the factor α directly influences the frequency of $\mathcal{Q}$ running empty and consequently the amount of computational complexity that is saved for search operations. If a large value is chosen for α , fewer messages will be added to $\mathcal{Q}$ and accordingly it will run empty more often. At the same time, however, if α is chosen too small, many messages with relatively small residuals will be updated, which conflicts with the main idea of IDS strategies.

LQRD can be seen as a generalization of RBP since it includes RBP as a special

case for $\alpha = 1$. In this case whenever $\mathbf{r}_{\max}$ is determined, only the message with the largest residual is appended to $\mathcal{Q}$, since there can be no message with $\mathbf{r}(c_{m,n}) > \alpha \cdot \mathbf{r}_{\max}$. After this message is updated, the queue will be empty and again the largest residual is determined. For $\alpha = 0$, on the other hand, every time $\mathbf{r}_{\max}$ is determined, all messages with residuals greater than zero are added to $\mathcal{Q}$ (note that messages in $\mathcal{Q}$ are in no specific order, though). In this case, LQRD is similar to (non-dynamic) message passing schedules that perform serial check node updates as presented, e.g., in [MS03, SLG04]. The only difference is that in each iteration the message that changes most is updated first and messages that do not change, i.e., have residual 0, are not updated at all. Hence, the factor α allows for a flexible adjustment of the computational overhead caused by the search for the largest residual – the smaller the value of α the smaller the overhead. At the same time, if α changes, the convergence speed and the error rate performance of LQRD change. In general, a higher value of α results in faster convergence. However, as α increases the algorithm becomes more and more greedy and a similar error floor behavior as for RBP is observed in the simulation results below.

Queue Residual Decoding (QRD)

The second proposed algorithm QRD is a modification of LQRD and has the potential of reducing the computational overhead even further, while providing similar results in terms of convergence speed and error rate performance. The difference between QRD and LQRD is that in the latter one, the message queue is generated once at the beginning of decoding and then only regenerated if it runs empty regardless of the values of the residuals that have been computed in the meantime (thus the name “lazy”). QRD, in contrast, allows for a message to be appended to $\mathcal{Q}$ as soon as its residual exceeds the threshold $\mathbf{r}_{\text{thresh}} = \alpha \cdot \mathbf{r}_{\max}$. This can be realized by simply checking whether a residual is greater than the current threshold whenever it is recomputed and if so, adding the corresponding message to $\mathcal{Q}$. Additionally, in QRD a message that has already been added to $\mathcal{Q}$ is not updated if its residual has dropped below the current threshold $\mathbf{r}_{\text{thresh}}$ since the time it has been appended. This more instantaneous handling of the message queue lets it run empty less frequently, which means that on average fewer search operations have to be executed per iteration. The exact description of QRD is given in Algorithm 13.

5.2.3 Simulation Results

In order to compare the performance of different dynamic schedules, the WiMAX (576, 288) rate-1/2 LDPC code is considered. The presented simulation results include six different decoding strategies:

- *BP, flooding*: the conventional BP algorithm with flooding schedule,
- *prob. shuffled*: BP decoding with the probabilistic shuffled schedule presented in Algorithm 9 in Section 5.1.1,
- *RBP*: the RBP algorithm from [CGW07a],

Algorithm 13 Queue Residual Decoding (QRD)

```

1: Initialize all  $v_{n,m} = L_n$ 
2: Initialize all  $c_{m,n} = 0$ 
3: Compute all  $\mathfrak{r}(c_{m,n})$ 
4: while stopping rule not satisfied do
5:   if  $\mathcal{Q}$  is empty then
6:     Find message with largest residual  $\mathfrak{r}_{\max} = \max_{\forall m,n} \mathfrak{r}(c_{m,n})$  and append it to  $\mathcal{Q}$ 
7:     Append all messages  $c_{m,n}$  with  $\mathfrak{r}(c_{m,n}) > \alpha \cdot \mathfrak{r}_{\max}$  to  $\mathcal{Q}$ 
8:   end if
9:   Remove message  $c_{m,n}$  from head of  $\mathcal{Q}$ 
10:  if  $\mathfrak{r}(c_{m,n}) > \alpha \cdot \mathfrak{r}_{\max}$  then
11:    Update message  $c_{m,n}$ 
12:    Set  $\mathfrak{r}(c_{m,n}) = 0$ 
13:    for  $m' \in \mathcal{M}(n) \setminus m$  do
14:      Update message  $v_{n,m'}$ 
15:      for  $n' \in \mathcal{N}(m') \setminus n$  do
16:        Compute  $\mathfrak{r}(c_{m',n'})$ 
17:        if  $c_{m',n'}$  not in  $\mathcal{Q}$  and  $\mathfrak{r}(c_{m',n'}) > \alpha \cdot \mathfrak{r}_{\max}$  then
18:          Append message  $c_{m',n'}$  to  $\mathcal{Q}$ 
19:        end if
20:      end for
21:    end for
22:  end if
23: end while

```

- *NWRBP*: the NWRBP algorithm from [CGW07a],
- *LQRD*: the novel presented IDS strategy from Algorithm 12,
- *QRD*: the novel presented IDS strategy from Algorithm 13.

Bit Error Rate and Convergence Speed

In Figure 5.3 the BER performance of LQRD and QRD for values of α between 0.0 and 1.0 in steps of 0.1 is shown after 20 equivalent iterations. While the complexity heavily depends on the choice of α , the BER performance is only influenced marginally in case of QRD and slightly more in the error floor region in case of LQRD. This allows for a flexible choice of α while almost maintaining the BER performance. Here, the best BER is achieved at $\alpha = 0.9$ for QRD and at $\alpha = 0.5$ for LQRD, but needs to be determined empirically for each code and each desired number of maximum iterations.

Figure 5.4 shows the BER as a function of E_b/N_0 after 5 (left) and 20 (right) equivalent iterations for all considered schedules, where $\alpha = 0.9$ is chosen for LQRD and $\alpha = 0.5$ for QRD. It can be seen that while NWRBP performs poorly for 5 iterations, RBP performs well but, in contrast to NWRBP, does hardly improve for a higher number of iterations. The performance of LQRD and QRD, however, is superior for both

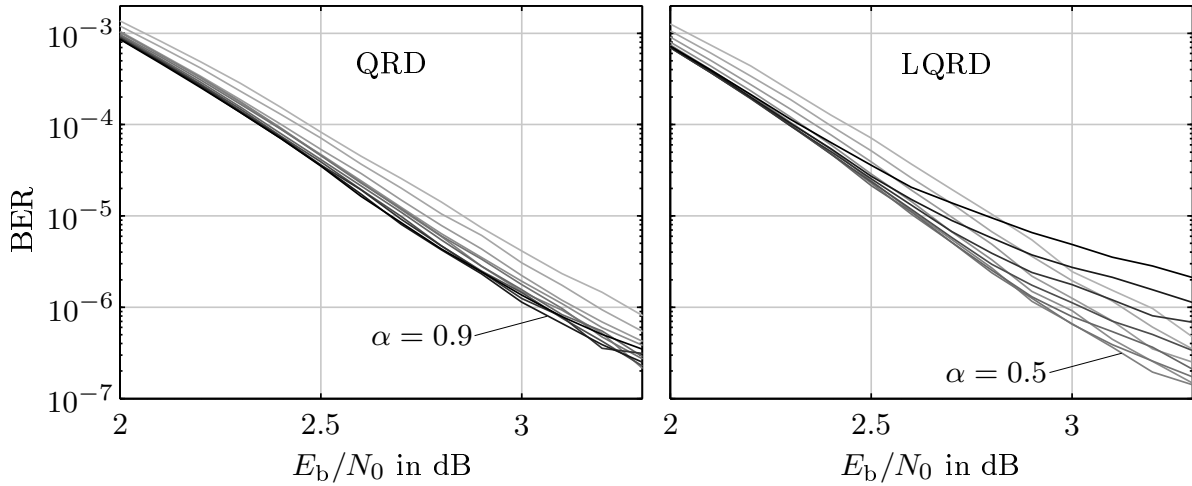


Figure 5.3: BER performance of QRD and LQRD for α between 0.1 (light) and 1.0 (dark) in steps of 0.1 after 20 equivalent iterations.

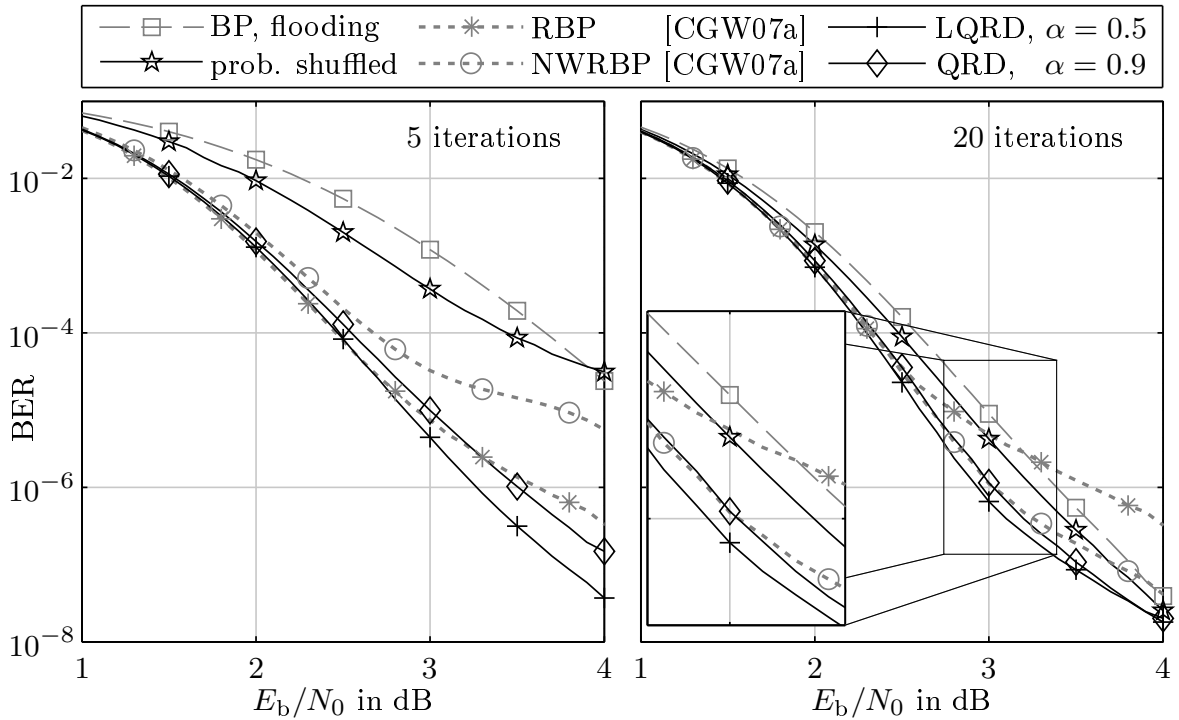


Figure 5.4: BER over E_b/N_0 after 5 (left) and 20 (right) equivalent iterations.

5 and 20 iterations and outperforms all other decoding strategies for all shown channel qualities including the probabilistic shuffled schedule proposed in Section 5.1.1. It can also be seen that for the chosen values of α , LQRD slightly outperforms QRD especially for smaller numbers of iterations.

Additionally, Figure 5.5 depicts the convergence properties of the different schedules. In the left plot, the BER over the number of equivalent iterations for $E_b/N_0 = 3$ dB is shown. The previous results are confirmed, as the convergence of QRD is almost as fast as that of RBP for smaller numbers of iterations while the performance roughly converges to that of NWRBP for higher numbers of iterations. LQRD on the other hands shows the fastest convergence and the best BER performance of all shown

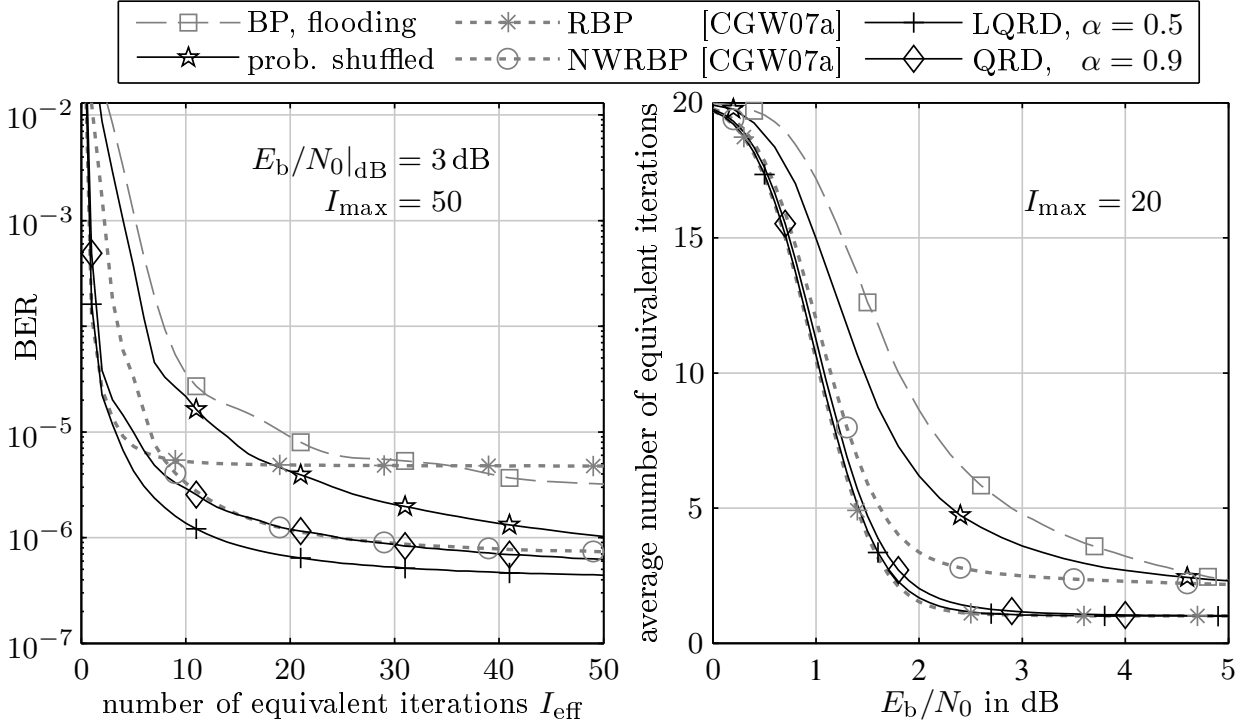


Figure 5.5: Left: BER over number of equivalent iterations at 3 dB. Right: average number of equivalent iterations over E_b/N_0 at $I_{\text{max}} = 20$.

algorithms and for all numbers of iterations.

The right plot in Figure 5.5 shows the average number of equivalent iterations as defined in section 5.2.1 over E_b/N_0 in dB. For LQRD, QRD, and RBP the smallest number of iterations has to be conducted over the whole range of channel qualities. This reduced number of iterations can be seen as partial remedy to the increased complexity per iteration of the IDS strategies as compared to non-dynamic schedules.

Complexity Comparison

As described in Section 5.2.2, LQRD and QRD are capable of reducing the additional complexity of IDS caused by the continuous search for the largest residual. In order to compare LQRD and QRD to RBP and NWRBP the average number of search operations performed per equivalent iteration are counted for the different schedules and for different values of E_b/N_0 .

In Figure 5.6 the average number of times the largest residual has to be determined per iteration is depicted as a function of E_b/N_0 (where a logarithmic scale is used for the y -axis). As explained in Section 5.2.1, the savings of NWRBP compared to RBP are in the order of the average check node degree $\bar{d}_c \approx 6.3$ of the employed WiMAX LDPC code. For LQRD and QRD curves are shown for values of $\alpha \in \{0.1, 0.5, 0.9, 1.0\}$ (where the curve for LQRD and $\alpha = 1.0$ corresponds to that of RBP). The setups LQRD with $\alpha = 0.5$ and QRD with $\alpha = 0.9$ are highlighted by markers since for these configurations the BER results are shown above. The number of search operations decreases significantly as α decreases. At $E_b/N_0|_{\text{dB}} = 1$ dB, e.g., for QRD with

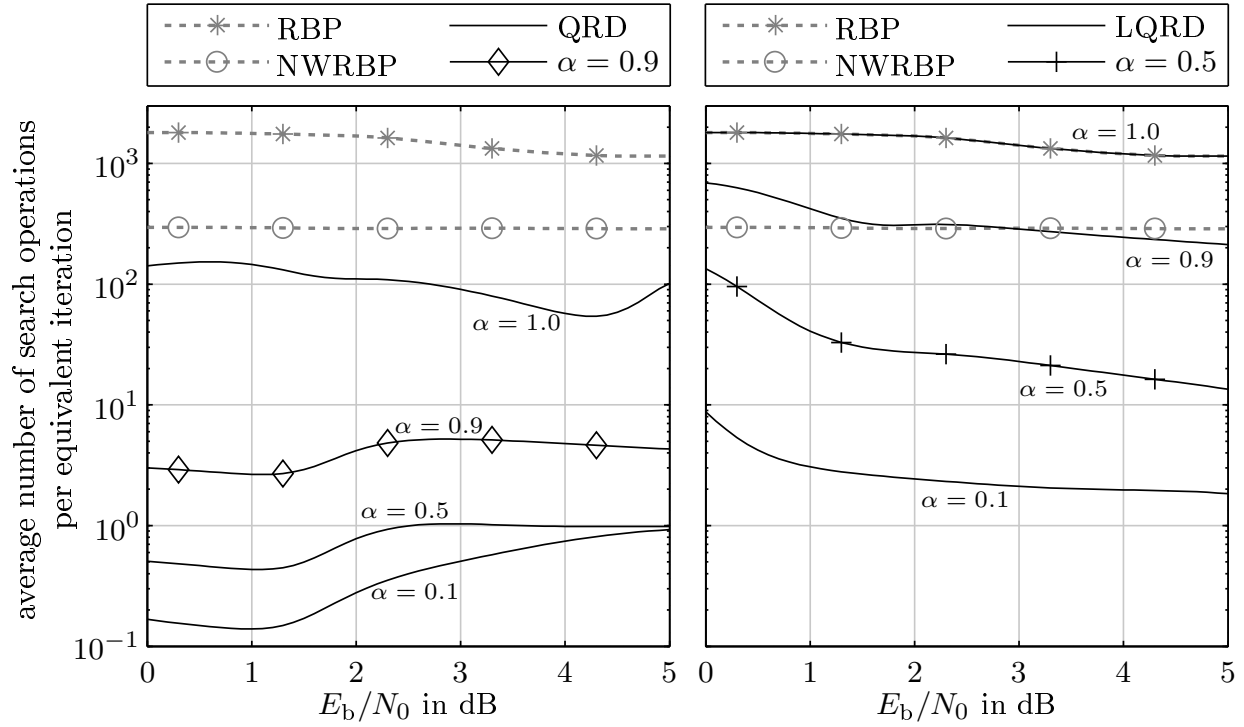


Figure 5.6: Average number of performed search operations per iteration needed for convergence over E_b/N_0 for QRD (left) and LQRD (right).

$\alpha = 0.1$ the number of search operations is reduced by a factor of up to 12000 compared to RBP and up to 2100 compared to NWRBP.

All things considered, LQRD and QRD are ultimately superior to the currently known IDS strategies RBP and NWRBP since they show equal – and in some cases even exceeding – performance while considerably reducing the additional complexity introduced by IDS.

5.3 Concluding Remarks

This chapter presents two novel types of decoder schedules: probabilistic sequential scheduling (Section 5.1) and (lazy) queue residual decoding (Section 5.2). These advanced schedules provide an increased flexibility in the choice of suitable decoding algorithms for devices with different requirements and computational capabilities.

The probabilistic sequential schedules are shown to combine the fast convergence of previous sequential schedules with improved error correction performance for both binary and non-binary LDPC codes. Since the additionally induced complexity is negligible, they are suited for low-complexity devices requiring small numbers of iterations and also delay-insensitive applications with low error rates requirements.

Even faster convergence and also lower error rates can be achieved by IDS strategies. The proposed queue residual decoding methods improve upon currently known IDS approaches in terms of both error correction performance and complexity while

maintaining the same fast convergence. They also outperform the probabilistic sequential schedules but generally exhibit a much higher per-iteration complexity than non-dynamic schedules. Consequently, they are particularly suitable for devices with high computational capabilities to accomplish very low decoding delay with best possible error correction performance.

All presented decoding algorithms are directly applicable to the multi-rate and multi-length LDPC codes presented in Chapters 3 and 4 as the decoders of these codes are based only on a single underlying mother code.

GPU Implementation of Non-Binary LDPC Decoding¹

The performance gain that can be achieved by using non-binary LDPC codes as introduced in Section 2.10 on pages 43 ff. comes at the cost of rapidly increasing decoding complexity. To deal with this increased complexity, an efficient implementation of a signed log domain FFT decoder for non-binary irregular LDPC codes was presented in [BMSV13, BMSV15], which exploits the inherent massive parallelization capabilities of message-passing decoders. Nvidia's *Compute Unified Device Architecture* (CUDA) was employed to incorporate the available processing power of state-of-the-art *Graphics Processing Units* (GPUs). Furthermore, a CUDA implementation of the signed log domain FFT decoder using the layered update rule [Hoc04], in which check nodes are updated one after another, was presented in [BMSV15]. This sequential updating of nodes has been shown to converge about twice as fast as the traditional flooding scheme. To achieve a high speedup of the layered CUDA implementation, quasi-cyclic non-binary LDPC codes are employed since they allow to update multiple neighboring check nodes in parallel without any performance loss. The most important concepts and results of this GPU implementation are presented in the following.

6.1 GPU Programming

The massive computational power of state-of-the-art graphic cards has led to the utilization of *Graphics Processing Units* (GPUs) for signal processing applications, e.g., [CC09], [NH09]. Graphic cards with GPUs by Nvidia Corporation can be elegantly programmed in a language similar to C using Nvidia's CUDA [Hal08]. Since more recently, GPUs from other manufacturers can also be utilized using the more general OpenCL specification [Khr]. The main advantage of GPUs is the large amount of processing cores available on the processor, even in low-end, low-cost products. These can be used for a parallelized execution of algorithms from numerous fields.

¹The descriptions in this chapter follow the ones given in [BMSV15].

The first work describing a generic (i.e., not tailored to a specific code) implementation of binary and non-binary LDPC codes is [Mon10]. This implementation of non-binary LDPC decoders based on the signed log domain FFT described in Section 2.10.2 has been presented in detail in [BMSV13]. Related work describing the implementation of non-binary LDPC decoders on GPUs are found in [AFSK13] and [WSY⁺12], where the FFT-based algorithm in the probability domain as introduced in [DM98a, DM98b] is employed, which suffers from numerical instabilities as compared to LLR based processing. The implementation presented here, on the other hand, is based on LLR arithmetic which has been shown to have appealing properties in practical implementations, e.g., the requirement of relatively short bit widths to represent the messages and an improved numerical stability. Furthermore, all multiplications in a probability domain implementation can be replaced by additions, further simplifying the realization. A GPU-based implementation of a shuffled non-binary LDPC decoder has been presented in [RC12]. Besides using computations in the probability domain, the authors of [RC12] realize a shuffled decoder from a variable node perspective [ZF02]. Although this shuffled decoder has a similar performance to the layered decoder [Hoc04] implemented here, the latter has several advantages related to the implementation, especially for higher rate codes. It is for instance one of the most widely used decoder implementations in fiber optic communications [LS13].

6.2 Log-FFT Belief Propagation for Non-Binary LDPC Codes

A massively parallelized implementation of the signed log domain FFT based decoder [BT04] as described in Section 2.10.2 is presented. The implementation is generic, i.e., it works for arbitrary field sizes q and irregular codes and does not require any special code structure. For an effective GPU implementation, two main points have to be considered: the arrangement of the data in the device memory and the use of the capabilities of the hardware for accelerating the data processing. An introduction to the CUDA terminology and specifics can be found in [Coo12].

6.2.1 Description of the Memory Arrangement

The most important structures are the messages $\mathbf{Q}_{n,m}$ and $\mathbf{R}_{m,n}$ that are passed between variable nodes and check nodes and the *a posteriori* values $\mathbf{Q}_n$. All of these are accessed very frequently in most *CUDA kernels* of the main loop (in CUDA terminology, a kernel denotes an enclosed subroutine). LDPC codes of different sizes and degree distributions as well as different Galois field dimensions should be supported, such that the potentially required amount of memory for these message vectors can become too large for on-chip memory. In addition, the need for quasi-random access to the message structures only leaves the global device memory as suitable solution.

Although the data is stored as a one-dimensional array in the device, it is interpreted as 2D ($\mathbf{Q}_n$) or 3D ($\mathbf{Q}_{n,m}$ and $\mathbf{R}_{m,n}$) structures, depending on the indexing as shown

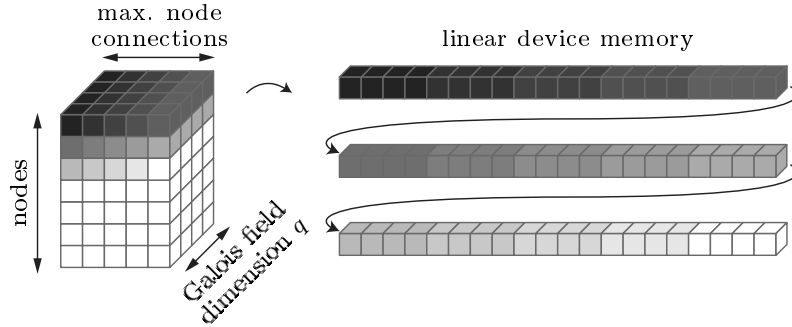


Figure 6.1: Three-dimensional structures used for message vectors and corresponding linear data allocation in the global device memory.

exemplarily for the 3D case in Figure 6.1. For the 3D arrays, the first index is pertinent to the node, the second to the node connection and the third to the Galois field element that corresponds to one entry of an LLR vector. The 2D arrays are indexed first by the node and second by the Galois field element. This indexing points to the exact position of one element in the structure. By arranging the data in this order, it is possible to access the memory in a coalesced way, such that multiple float values are handled in only one read/write memory access. If each of the q adjacent LLR vector entries are accessed by q adjacent threads, this access will be coalesced, which is a critical requirement for achieving high throughput performance.² This arrangement boosts the performance with increasing q . CUDA requires fixed sizes for data structures, and since irregular LDPC codes do not have a constant number of Tanner graph edges per node, it is obligatory to set the node connection dimension to the maximum value of all node degrees, taking care of not accessing the invalid positions during the processing. Temporary data that is needed within the kernels is either stored in fast on-chip registers or in the on-chip shared memory if the data is required by multiple threads. The shared memory is generally faster than the global memory but only available to the threads within one *block of threads*, which denotes a unit that is guaranteed to be executed on the same processing unit by the CUDA architecture. A more extensive description of the employed data structures can be found in [Mon10].

Another GPU-specific implementation detail is the employment of the GPU's *texture memory* for the Galois field arithmetic. Two-dimensional lookup tables of size $q \times q$ for the addition and multiplication of Galois field elements as well as one-dimensional lookup tables for the inversion of Galois field elements (size $q - 1$) and the conversion between exponential and decimal representation (size q) are initialized in the texture memory and are thus available to all kernels.

6.2.2 Description of the CUDA Kernels

The implementation follows the algorithm described in Section 2.10.2 and is depicted in Figure 6.2. Each of the gray shaded blocks corresponds to one CUDA kernel.

²This feature is characteristic for GPU architectures and similar to the SIMD (single instruction multiple data) concept, in which the same operation is performed on multiple input data elements by multiple processing units in parallel.

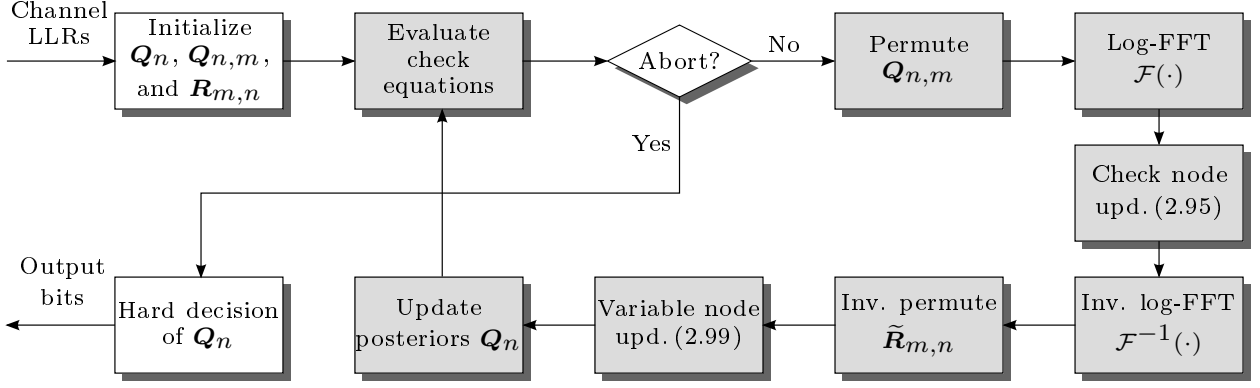


Figure 6.2: Flowchart of the signed log domain FFT BP decoding algorithm. Each gray shaded block corresponds to one CUDA kernel.

Initially, the number of iterations is zero. After initialization, the input data is transferred to the CUDA device for an initial check of the parity equations. If they are not fulfilled, the main decoding loop is executed until they are, or until a maximum number of iterations is reached. When the iterative decoding process finishes, a hard decision is conducted to obtain the final binary decoding result. In the following, the most important aspects of the main loop's CUDA kernel implementations are explained.

Permute Message Vectors $Q_{n,m}$ (2.92) and $\tilde{R}_{m,n}$ (2.98)

Before and after the check node update, the elements of the message LLR-vectors have to be permuted according to the Galois field element at the corresponding check matrix position (cf. Section 2.10 and [BT04]). A shared memory block is used as temporary memory, allowing a parallel implementation of the permute operation. For permuting $\tilde{R}_{m,n}$ according to (2.98), the shared memory stores the input data using the permuted indices according to the matrix entry $H_{m,n}$ (using the texture tables for multiplication), and writes the output data directly. For the permutation of $Q_{n,m}$ according to (2.92) the shared memory reads the input values directly, but the output is written using the permuted indices as shown in Figure 6.3. This way, one thread (i.e., one of the many processing units of the GPU) per node, per node connection, and per Galois field element can be employed. The total number of threads for this kernel

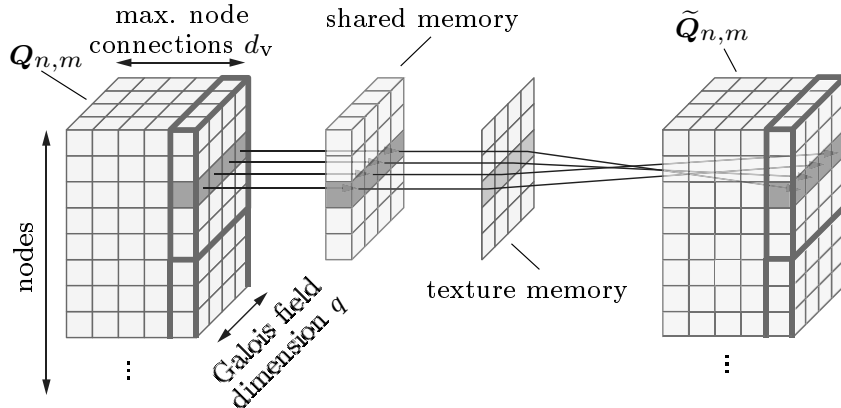


Figure 6.3: Permutation of message vectors $Q_{n,m}$.

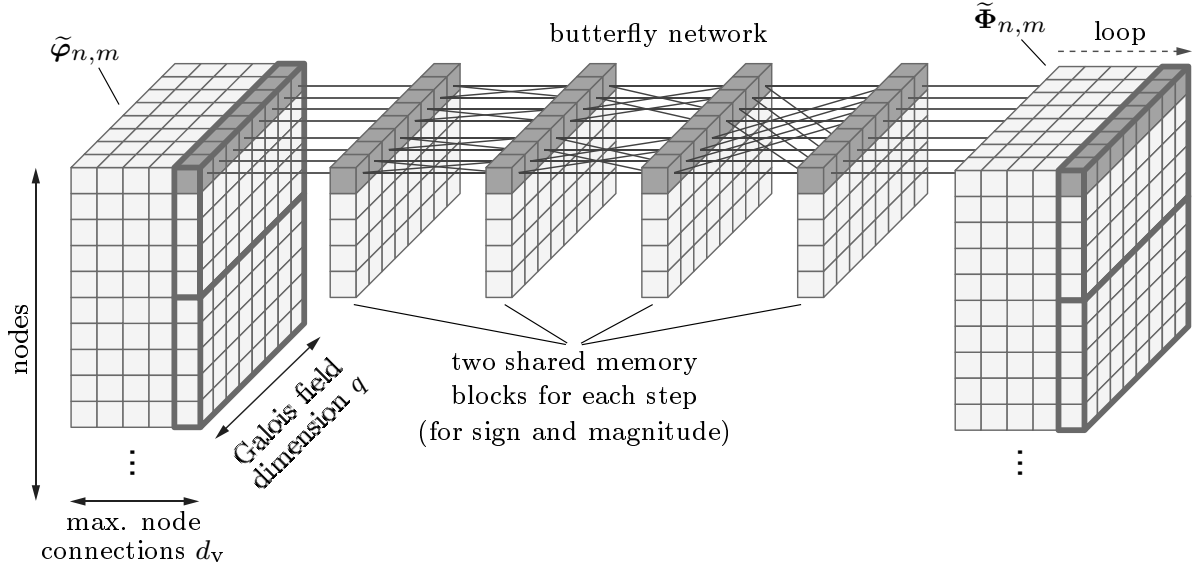


Figure 6.4: Butterfly network for FFT implementation.

is $N_q \cdot d_v \cdot q$ for the permutation of $\mathbf{Q}_{n,m}$ and $M_q \cdot d_c \cdot q$ for the inverse permutation of $\tilde{\mathbf{R}}_{m,n}$, respectively, where d_v denotes the maximum variable node degree and d_c denotes the maximum check node degree.

Signed Log Domain FFT of $\tilde{\varphi}_{n,m}$ (2.94) and $\tilde{\Theta}_{m,n}$ (2.96)

The implementations of the signed log FFT and the inverse signed log FFT are identical and follow a butterfly diagram as shown in Figure 6.4. As explained in [BT04], additions in the log domain consist of two separate operations for the magnitude and the sign, which is why two shared memory blocks are required for this purpose (i.e., each shared memory block depicted in Figure 6.4 actually consists of two separate equally sized blocks). Both operations are executed simultaneously in the CUDA kernel implementation. The signed log FFT is applied independently to each LLR-vector of size q . Thus, the CUDA block width is fixed to q . For each node and each Galois field element, one thread processes all node connections in a loop to compute the signed log FFT. The execution of the butterfly is supervised by a small loop of $\log_2 q$ iterations.

The exponential and logarithm functions that are needed for the signed log domain operations [BT04] are realized by the fast hardware implemented functions which are provided by the GPU. The transfer of data between the global memory and the shared memory is done in a coalesced way, which is ensured by proper indexing.

Check Node Update of $\tilde{\Theta}_{m,n}$ (2.95)

The implementation is visualized in Figure 6.5 and uses one thread per node and Galois field element to compute the outgoing message LLR-vectors for each edge of the corresponding check node. Since the computations are performed in the signed log domain, magnitude and sign computations are performed separately. In each thread, a first loop over the node's edges calculates the total sum of the magnitude values

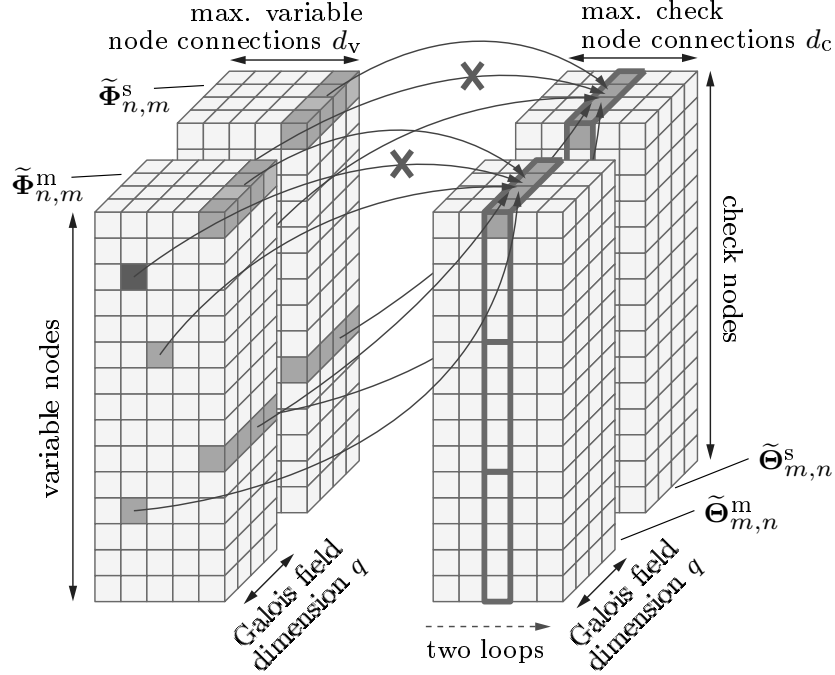


Figure 6.5: Check node update of message vectors $\tilde{\Theta}_{m,n} = \left(\tilde{\Theta}_{m,n}^s, \tilde{\Theta}_{m,n}^m \right)$.

and the total product of the sign values. In a second loop, the current magnitude value is subtracted from the total magnitude sum to acquire the extrinsic sum, and the total sign is multiplied by the current sign value for each edge. Shared memory is not required since all memory accesses are executed in a coalesced fashion and, thus, provide high throughput performance.

Variable Node Update of $Q_{n,m}$ (2.99)

The kernel processes one thread per variable node and Galois field dimension (see Figure 6.6). An internal loop processes all the node connections within one thread. Each thread first reads the input LLR L_n into a shared memory block and then adds all extrinsic information from $R_{m,n}$ for the current edge. In a second shared memory block, the maximum resulting element $\beta_{n,m}$ of the LLR-vector is found and subtracted from each entry from the first shared memory block before the result is written into $Q_{n,m}$.

Update A Posteriori Values Q_n (2.100)

The implementation of this kernel follows the one for the variable node update, except that the search and subtraction of the maximum element $\beta_{n,m}$ is omitted and the information of all edges (rather than all but one) is added to the input LLR to yield the *a posteriori* LLR-vectors Q_n .

Evaluate Check Equations

First, one thread per variable node is employed to determine a hard decision of the node's *a posteriori* LLR-vector by looping over the Galois field dimension to find

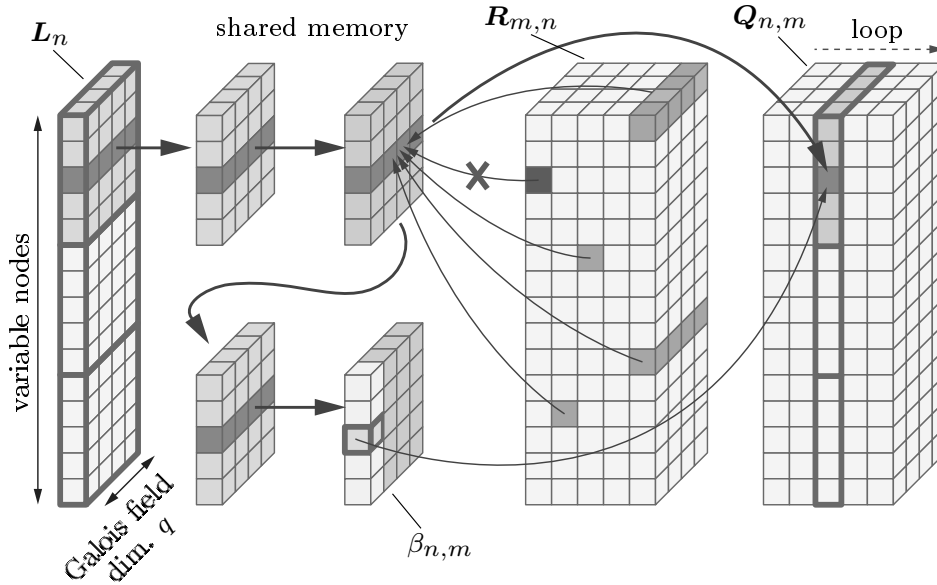


Figure 6.6: Variable node update of message vectors $Q_{n,m}$.

the maximum element. After this, one thread per check node loops over its edges and utilizes the texture tables for Galois field addition and multiplication to add up the hard decision values. If and only if the result is zero, the check equation is fulfilled, resulting in a boolean vector of length M_q . To decide if decoding can be stopped, a parallelized reduction algorithm is applied to sum up the (0 or 1) elements of this vector in $\log_2 M$ steps (over the natural numbers). The parity equations are fully satisfied if the result of the sum is zero (i.e., all boolean values are zero).

6.3 Layered Decoding for Non-Binary QC LDPC Codes

The CUDA implementation presented above was also extended by the layered check node update rule as proposed for decoding of binary LDPC codes in [Hoc04]. To enable a high speedup for this inherently sequential schedule, QC LDPC codes are employed. Virtually all LDPC codes used in practical systems today belong to this class and allow for a partial parallelization of the sequential decoder. The CUDA framework is employed to adapt a single-core reference CPU implementation to be executed on one of Nvidia's general purpose GPUs. Decoder throughput results are presented for the CPU as well as the GPU implementations for different Galois field dimensions.

In the following, the layered schedule for the log-FFT BP algorithm is introduced, and the most important modifications that were required for its implementation will be explained.

6.3.1 Log-FFT Belief Propagation with Layered Schedule

The *layered decoding* or *row-decoding* as introduced in [Hoc04] is particularly relevant for the decoding of practical codes, as it allows an increase in the speed of convergence of the iterative decoder by about 50% (depending on the code). This means that the number of required decoding iterations can be significantly reduced compared to the flooding implementation described in Section 6.2. Especially in the case of quasi-cyclic LDPC codes, the layered decoder is an attractive decoding architecture, as it allows realizing a certain level of parallelism with good communication performance at relatively low hardware costs.

The main loop of the layered decoder consists of three main steps. First, a simple preparation step generates the check node input data $\mathbf{T}_{n,m}$. Then, the check node update (including permutation and log-FFT transform) and thus the generation of new extrinsic data $\mathbf{R}_{m,n}$ is executed. Finally, the *a posteriori* memory $\mathbf{Q}_n$ is updated. The decoder carries out these three steps sequentially for each row (or group of rows) of the parity check matrix $\mathbf{H}$. After all rows have been considered, a single decoding iteration has been carried out. Starting from the received channel values $\mathbf{L}_n$, the layered decoder continuously updates an estimate of the *a posteriori* values $\mathbf{Q}_n$. It should be noted that in contrast to the implementation of the flooding schedule, there is no explicit variable node update.

In the following, the three steps for a single row m of $\mathbf{H}$ are described. The first preparation step consists in computing $|\mathcal{N}(m)|$ temporary message vectors

$$\mathbf{T}_{m,n} = \mathbf{Q}_n - \mathbf{R}_{m,n}, \quad \forall n \in \mathcal{N}(m) \quad (6.1)$$

for all non-zero entries (indexed by n) of the m -th row of $\mathbf{H}$. At the beginning of the decoding of a single frame, all $\mathbf{R}_{m,n}$ are initialized by zero and are then continuously updated.

In the second step of the layered schedule, the actual decoding takes place. The temporary values $\mathbf{T}_{m,n}$, corresponding to the messages at the inbound edges of check nodes, are used to compute the check node update. First, the $|\mathcal{N}(m)|$ elements of $\mathbf{T}_{m,n}$ are permuted according to the matrix entry $a = H_{m,n} \in \mathbb{F}_q^+$ with

$$\tilde{\mathbf{T}}_{m,n} = \mathbf{T}_{m,n} \cdot \mathbf{P}_a \quad (6.2)$$

where the $q - 1$ permutation matrices $\mathbf{P}_a$ correspond to the ones introduced in Section 6.2 and (2.92). Following the permutation, the values $\tilde{\mathbf{T}}_{m,n}$ are transformed into the signed log domain according to $\tilde{\boldsymbol{\varphi}}_{n,m} = (\tilde{\boldsymbol{\varphi}}_{n,m}^s, \tilde{\boldsymbol{\varphi}}_{n,m}^m) = (\mathbf{1}, \tilde{\mathbf{T}}_{m,n})$. Following the application of the FFT $\mathcal{F}(\cdot)$, the check node update according to (2.95a) and (2.95b), and the inverse transform $\mathcal{F}^{-1}(\cdot)$ yielding $\tilde{\boldsymbol{\theta}}_{m,n} = (\tilde{\boldsymbol{\theta}}_{m,n}^s, \tilde{\boldsymbol{\theta}}_{m,n}^m)$ and $\tilde{\mathbf{R}}_{m,n} = \tilde{\boldsymbol{\theta}}_{m,n}^m$, the output information of the check node update can be computed as

$$\mathbf{R}_{m,n} \stackrel{(2.98)}{=} \tilde{\mathbf{R}}_{m,n} \cdot \mathbf{P}_a^{-1}.$$

Finally, in the last step of the processing for row m , the *a posteriori* memory is updated according to

$$\mathbf{Q}_n = \mathbf{T}_{m,n} + \mathbf{R}_{m,n}, \quad \forall n \in \mathcal{N}(m). \quad (6.3)$$

This last step concludes the update of a single check equation and row m , and the decoder can continue with decoding the next row (or group of rows).

Using the layered schedule, the memory requirements are also significantly reduced compared to the flooding strategy. Only a single type of edge messages must be stored in the memory ($\mathbf{R}_{m,n}$), and the decoder basically continuously updates *a posteriori* values $\mathbf{Q}_n$.

In this general description of the layered schedule, the different *a posteriori* memory locations are updated several times per iteration (depending on the variable node degree of the parity-check matrix), each time with newly refined extrinsic information from the associated check node update. This means that in the most general case, the three operations described above need to be carried out in a sequential manner. This sequential execution and the use of refined information is also responsible for the good convergence behavior of the layered schedule. If, however, QC LDPC codes as introduced in Section 2.9 are employed, the independence of the Z check nodes in each of the M'_q groups can be exploited to parallelize the preparation step, the check node update step, and the *a posteriori* update step. A flowchart of the layered decoding algorithm tailored to QC LDPC codes is shown in Figure 6.7.

6.3.2 Description of the Memory Arrangement

As described in the previous section, the layered decoder only needs to store the check node messages $\mathbf{R}_{m,n}$ and the *a posteriori* values $\mathbf{Q}_n$ from one iteration to the next. As before, due to their potential size, these structures are stored as 3D ($\mathbf{R}_{m,n}$) and 2D ($\mathbf{Q}_n$) structures in the global device memory. Since in each inner iteration, the messages of only Z check nodes are updated, the temporary three dimensional memory for the LLR message vectors only needs to be of size $Z \cdot d_c \cdot q$. Thus, the memory requirements for this temporary memory is reduced by a factor of $M'_q = M_q/Z$.

6.3.3 Description of the CUDA Kernels

The implementation follows the description in Section 6.3.1 and the flowchart given in Figure 6.7, where again gray shaded blocks represent CUDA kernels.

After initialization of the *a posteriori* values, the check equations are evaluated according to Section 6.2.2 to check if a valid code word has been received. If they are not fulfilled, the main decoding routine is entered. It consists of an outer loop for the decoding iterations and an inner loop which sequentially processes groups of Z check nodes (and thus performs $M'_q = M_q/Z$ passes in each outer iteration). The inner iteration counter is denoted c and indicates that check nodes with indices $(c-1)Z + 1$

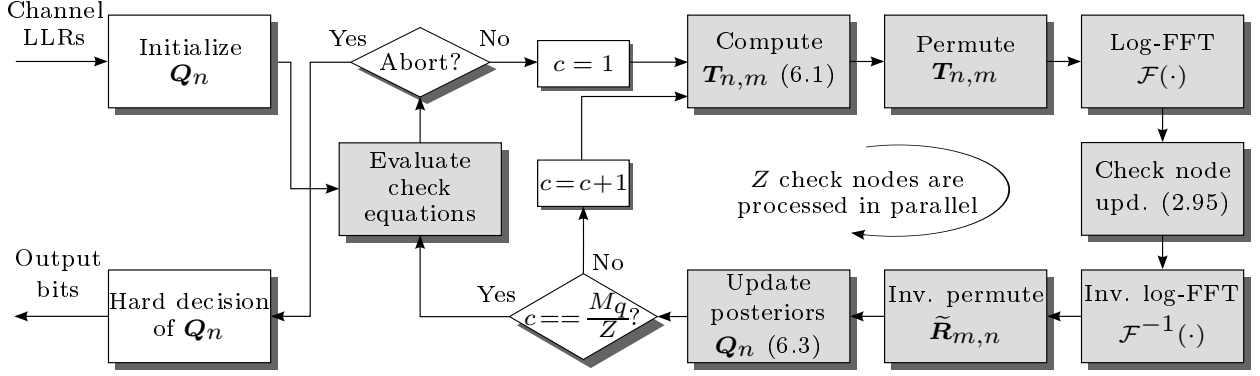


Figure 6.7: Flowchart of the signed log domain FFT BP decoder with layered schedule. Each gray shaded block corresponds to one CUDA kernel.

to cZ are currently processed. The only input to each pass of the inner loop are the current *a posteriori* values $\mathbf{Q}_n$ and the check-to-variable messages $\mathbf{R}_{m,n}$ of check nodes $(c-1)Z+1$ to cZ of the last outer iteration. The output of each pass of the inner loop are updated *a posteriori* values $\mathbf{Q}_n$ and the updated check-to-variable messages $\mathbf{R}_{m,n}$ of check nodes $(c-1)Z+1$ to cZ . While each check-to-variable message is computed exactly once during an outer iteration, the *a posteriori* values at each variable node n are updated $d_{v,n}$ times, where $d_{v,n}$ denotes the degree of variable node n , always using the most up-to-date check node information. Sequentially updating the *a posteriori* values is the reason for the layered schedule to converge faster than the traditional flooding schedule.

In the following, each CUDA kernel of the inner loop is explained.

Compute Extrinsic Check Node Input $\mathbf{T}_{m,n}$ (6.1)

Instead of having a variable node update to compute the variable-to-check messages, the extrinsic check node input is computed for each of the Z check nodes in the current group c at the beginning of each inner iteration from the *a posteriori* values and the previous check-to-variable message. For this, one thread per check node in the current group c and per Galois field element (i.e., $Z \cdot q$ threads) is initiated and computes the extrinsic information at each incoming edge in a loop according to $\mathbf{T}_{m,n} \stackrel{(6.1)}{=} \mathbf{Q}_n - \mathbf{R}_{m,n}$ for $(c-1)Z+1 \leq m \leq cZ$.

Permute Message Vectors / Signed Log Domain FFT / Check Node Update

These three kernels are identical to the ones in the implementation of the flooding schedule. Since only Z check nodes can be processed in parallel, however, only $Z \cdot d_c \cdot q$ threads are initiated as opposed to $M_q \cdot d_c \cdot q$ in the flooding schedule, resulting in a reduction of $M'_q = M_q/Z$ in the total number of threads.

Update of A Posteriori Values $\mathbf{Q}_n$ (6.3)

While having the same purpose, this kernel is different from the corresponding implementation for the flooding schedule. In the latter one, all check-to-variable messages

are computed in parallel, such that afterwards all incoming messages of a variable node are summed up (together with the channel LLR vector) to yield the *a posteriori* LLR vector. During one iteration of the layered schedule, however, check nodes are updated sequentially and, thus, the *a posteriori* LLR vectors are updated multiple times, since each variable node is connected to multiple check nodes. Due to the structure of QC LDPC codes, in a group of Z check nodes that are processed in parallel, no two check nodes are connected to the same variable node. As a result, in one inner iteration, no *a posteriori* LLR vector is affected by more than one check node update. Consequently, this allows all affected *a posteriori* values to be updated in parallel without any memory conflicts. To achieve this, one thread for each of the Z current check nodes and for each Galois field element (i.e., $Z \cdot q$) is used to update the *a posteriori* values with respect to the updated check-to-variable messages according to $\mathbf{Q}_n \stackrel{(6.3)}{=} \mathbf{T}_{m,n} + \mathbf{R}_{m,n}$ for $(c-1)Z + 1 \leq m \leq cZ$. The different edges of each check node are processed by each thread in a loop.

6.4 Performance Evaluation

Results for the flooding schedule and the layered schedule are presented separately in Sections 6.4.1 and 6.4.2, respectively.

6.4.1 Flooding Schedule

The CUDA implementation of the flooding schedule is executed under the same settings as a single-threaded CPU reference implementation, and the decoder throughput is measured at several E_b/N_0 values and for codes over different Galois fields $\mathbb{F}_q$, with $q = 2^p$ and p between 1 and 8. It should be noted that the obtained results are related to the hardware architecture used in the simulations and will differ if different CPUs and/or GPUs are used. Here, the following standard hardware was used:

- CPU: Intel Core i7-920, 2.67GHz, 6GB RAM
- GPU: GeForce GTX 580, 512 CUDA cores, 1536 MB RAM, CUDA Compute Capability 2.0.

Table 6.1 shows the different non-binary rate- $\frac{1}{2}$ LDPC codes used for obtaining the results in this section. Since one Galois field symbol is composed of p bits, the code size N_q is always chosen according to $N_q = N/p$, such that the equivalent number N of bits per code word is constant. Concentrated check node degree distributions are used for all codes. The non-binary check matrices are constructed with the PEG algorithm [HEA05], by first constructing a binary matrix of the given dimensions and subsequently replacing all ones by non-zero Galois field elements chosen uniformly at random. A maximum number of $I_{\max} = 100$ decoding iterations are conducted. For all employed codes, the error rate results of the GPU and the CPU implementation are identical.

$q = 2^p$	$N_q = \frac{N}{p}$	M_q	Λ_2	Λ_3	Λ_4	Λ_5
2	2304	1152	0.549	0.144	0.001	0.306
4	1152	576	0.537	0.302	0.042	0.119
8	768	384	0.643	0.150	0.194	0.013
16	576	288	0.773	0.102	0.115	0.010
32	462	231	0.848	0.143	0.009	0
64	384	192	0.940	0.050	0.010	0
128	330	165	0.852	0	0.148	0
256	288	144	0.990	0.010	0	0

Table 6.1: Dimensions and variable node degree distributions of employed non-binary rate- $\frac{1}{2}$ LDPC codes. Degree distributions for $2^p \in \{8, 16, 32, 64\}$ are taken from [HEA05].

In Figure 6.8 the average total throughput of (coded) bits that is processed by the CPU reference implementation (dashed lines) as well as the GPU implementation (solid lines) is depicted over the channel quality E_b/N_0 in dB. The bottom plot shows the curves for the CPU implementation in more detail. The very low total throughput at E_b/N_0 values below 1 dB is due to the high maximum number of 100 decoding

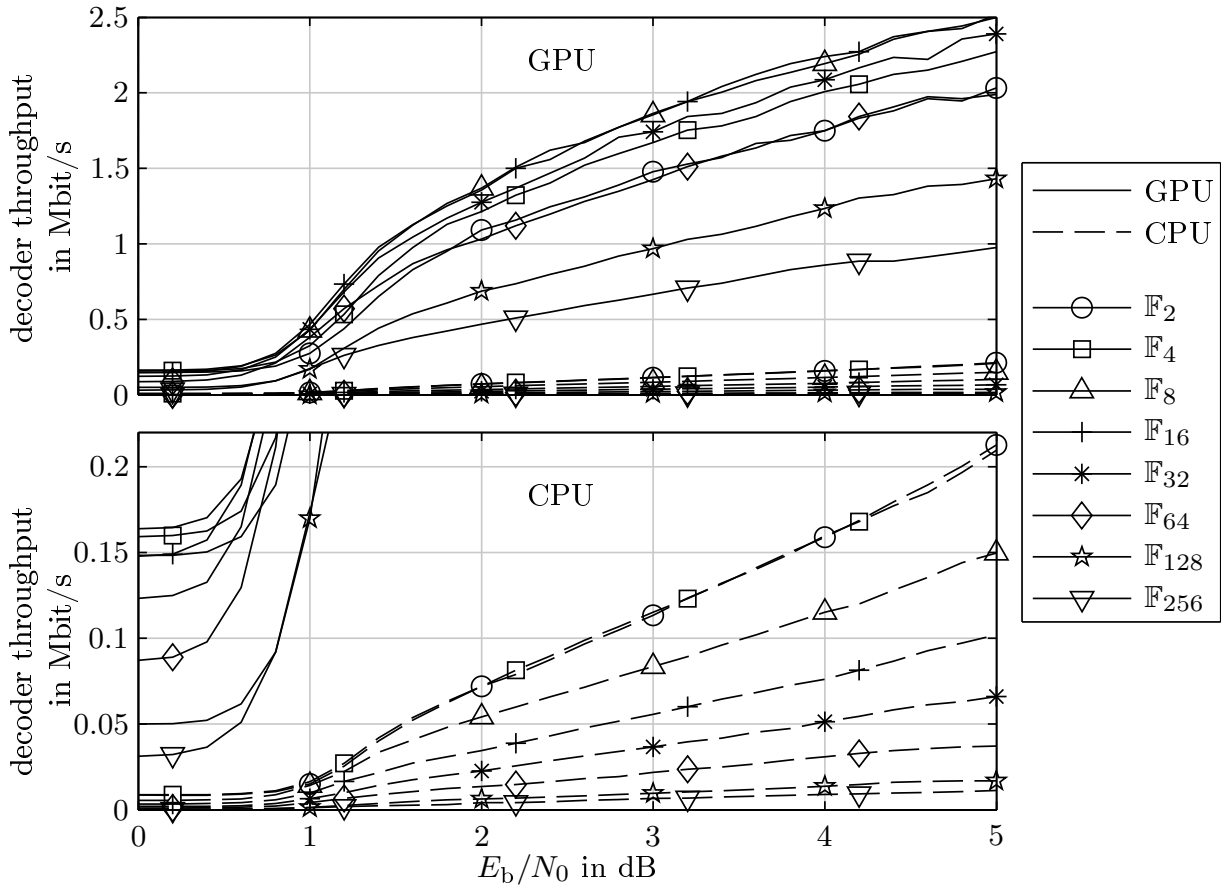


Figure 6.8: Average total decoder throughput of coded bits over E_b/N_0 for the flooding schedule (Section 6.2) at a maximum of $I_{\max} = 100$ iterations.

$E_b/N_0 _{\text{dB}}$	$\mathbb{F}_2$	$\mathbb{F}_4$	$\mathbb{F}_8$	$\mathbb{F}_{16}$	$\mathbb{F}_{32}$	$\mathbb{F}_{64}$	$\mathbb{F}_{128}$	$\mathbb{F}_{256}$
0 dB	17.2	18.0	29.6	43.9	61.1	83.6	112	128
1 dB	18.3	19.8	29.9	47.2	65.9	92.1	124	148
2 dB	15.2	16.8	25.2	39.2	56.2	75.8	109	113
3 dB	13.0	14.5	22.2	33.5	47.5	65.2	100	100

Table 6.2: Achieved Speedup $\frac{\text{throughput GPU}}{\text{throughput CPU}}$ of flooding schedule.

iterations that are almost always executed, before the waterfall region of the code's error rate curve is reached. For higher E_b/N_0 values, the total throughput steadily increases, since fewer decoding iterations have to be executed until a correct code word is recovered.

The speedup achieved by the GPU implementation with respect to the CPU implementation is shown in Table 6.2 for several E_b/N_0 values. The speedup hugely increases from below 20 to almost 150 at 1 dB with increasing Galois field dimension. The increasing speedup is mainly due to the fact that groups of q Galois field elements are stored in continuous memory positions, which leads to increased efficiency of the coalesced memory access. This increasing efficiency of the GPU implementation leads to the interesting effect that the average throughput is not monotonically decreasing with the Galois field dimension (cf. Figure 6.9), even though the decoding complexity increases as $\mathcal{O}(N2^p)$.

In Figure 6.9 the throughput results from above are plotted against the number of bits p per Galois field symbol for different E_b/N_0 values. Again, a zoomed region is shown for the CPU implementation. As expected, the throughput of the CPU implementation quickly decreases with p . The GPU implementation, on the other

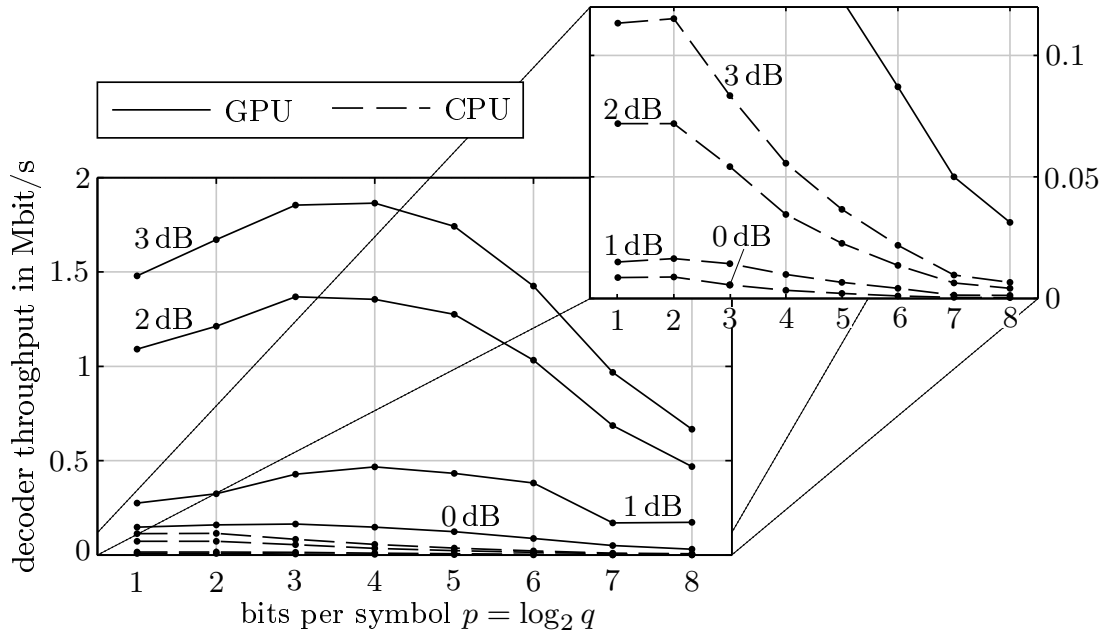


Figure 6.9: Throughput over Galois field dimension of the flooding schedule for $E_b/N_0|_{\text{dB}} \in \{0 \text{ dB}, 1 \text{ dB}, 2 \text{ dB}, 3 \text{ dB}\}$.

hand, shows an increasing throughput up to around 4 bits per symbol before it also starts to decrease. This behavior can be explained by the fact that, for moderate Galois field dimensions, the increasing computational complexity is overcompensated by the more efficient use of the GPU's capabilities due to the coalesced memory access of q addresses at once. However, for larger Galois field dimensions $q \geq 2^5$ a saturation occurs and the throughput starts to decrease.

6.4.2 Layered Schedule

The hardware setup for the evaluation of the CUDA implementation of the layered schedule is the same as for the flooding setup in Section 6.4.1. As mentioned before, QC LDPC codes are used to allow a high speedup of the layered schedule. For Galois field dimensions $q \in \{2, 4, 16, 64, 256\}$, lifting matrices $\mathbf{A}$ of rate 1/2 are constructed with the PEG algorithm using the optimized degree distributions from Table 6.1. The lifting factor is chosen as $Z = 32$ and the size of the lifting matrices are chosen as $N'_q \in \{96, 48, 24, 16, 12\}$ to result in an equivalent binary block size of $N = p \cdot N'_q \cdot Z = 3072$. The non-zero elements in each resulting parity check matrix are replaced by random non-zero elements of the respective Galois field. The maximum number of decoding iterations is set to only $I_{\max} = 10$ to emphasize the difference between the slowly converging flooding schedule and the layered schedule, which shows much faster convergence.

The BER results for $q \in \{2, 16, 256\}$ shown in Figure 6.10 clearly confirm the increased convergence speed of the layered schedule as compared to the flooding schedule. After 10 iterations and at BER 10^{-4} , the layered schedule outperforms the parallel schedule by about 0.5 dB for $q = 2$, by about 0.4 dB for $q = 16$, and by about 0.25 dB for $q = 256$. The BER results also illustrate the performance gain that is achieved by

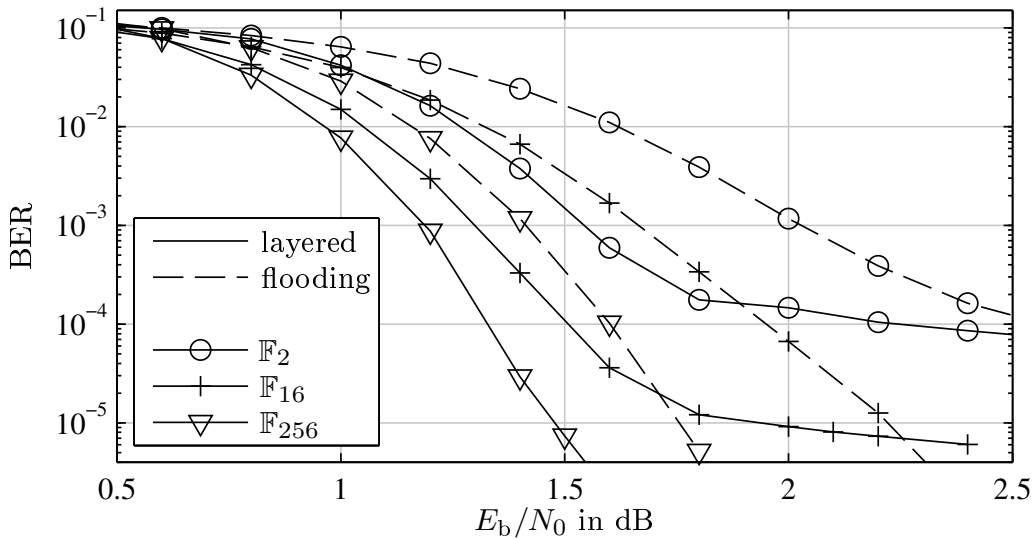


Figure 6.10: BER over E_b/N_0 of layered and flooding schedules for $q \in \{2, 16, 256\}$ and $I_{\max} = 10$ decoding iterations.

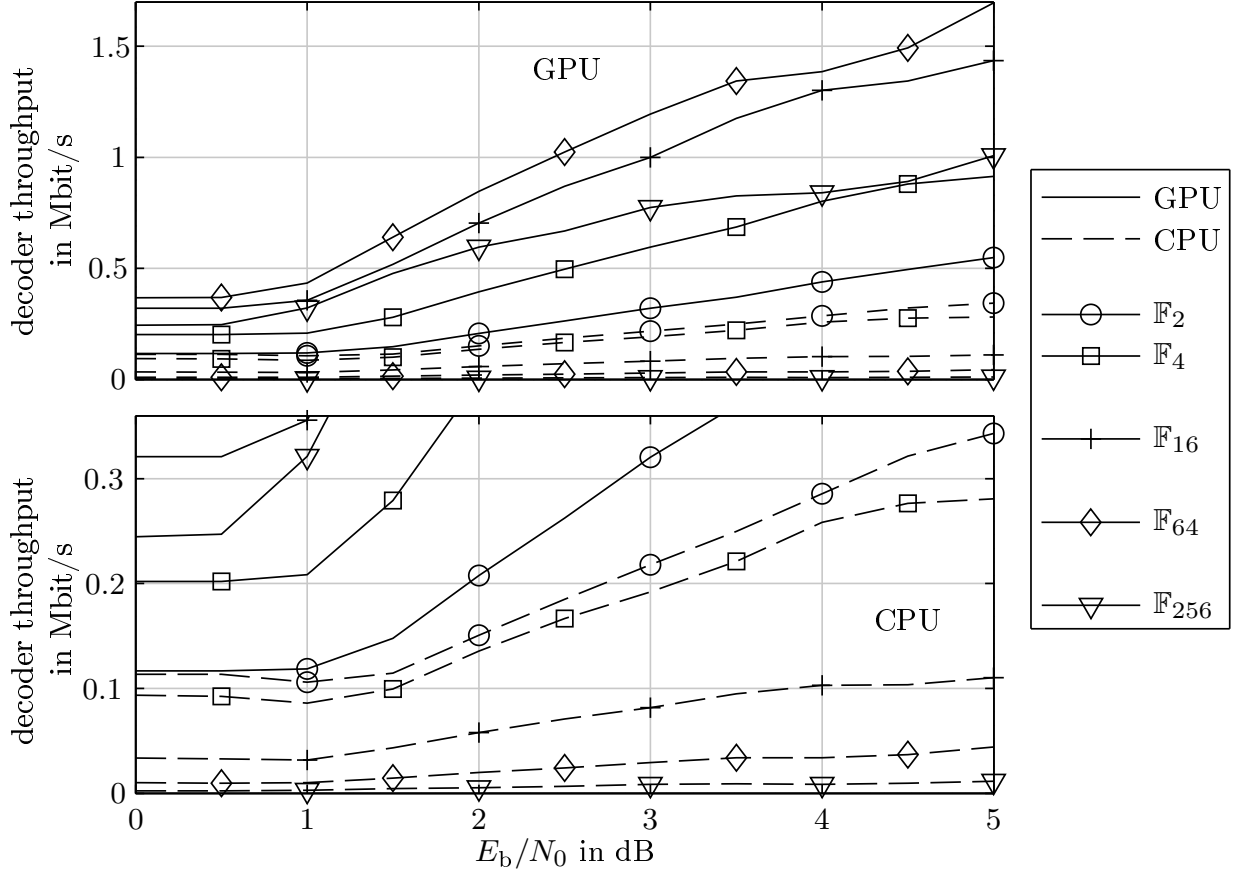


Figure 6.11: Average decoder throughput of coded bits over E_b/N_0 of the layered schedule at a maximum of $I_{\max} = 10$ iterations.

increasing the Galois field dimension. The $\mathbb{F}_{16}$ code outperforms the binary code by about 0.28 dB and the $\mathbb{F}_{256}$ code improves the BER performance by another 0.13 dB.

Figure 6.11 shows the average total throughput of code bits processed by the single-threaded CPU implementation (dashed lines) as well as the GPU implementation (solid lines) over E_b/N_0 . Like for the flooding schedule, throughput values increase with increasing E_b/N_0 since fewer decoding iterations have to be executed. While the CPU implementation exhibits best throughput performance for the binary code, the layered GPU implementation attains the highest throughput for the code over $\mathbb{F}_{64}$.

This observation is confirmed by Figure 6.12 in which the throughput is plotted against the number of bits per symbol p for different fixed E_b/N_0 values. A zoomed region is shown for the CPU implementation. Similar to the flooding schedule in Figure 6.9, the curves of the CPU implementation are steadily decreasing with increasing Galois field dimension. For the GPU implementation, however, the curves of the layered schedule achieve a maximum at $q = 64$, whereas this maximum was located between $p = 3$ and $p = 4$ for the flooding schedule. This behavior is explained by the larger number of threads that are initiated for each kernel for the flooding schedule ($M_q \cdot q$) as compared to the layered schedule ($Z \cdot q = q \cdot M_q / M'_q$). To launch a sufficient number of threads – and thus fully exploit the compute capability of the GPU – a higher Galois field dimension q is required for the layered schedule.

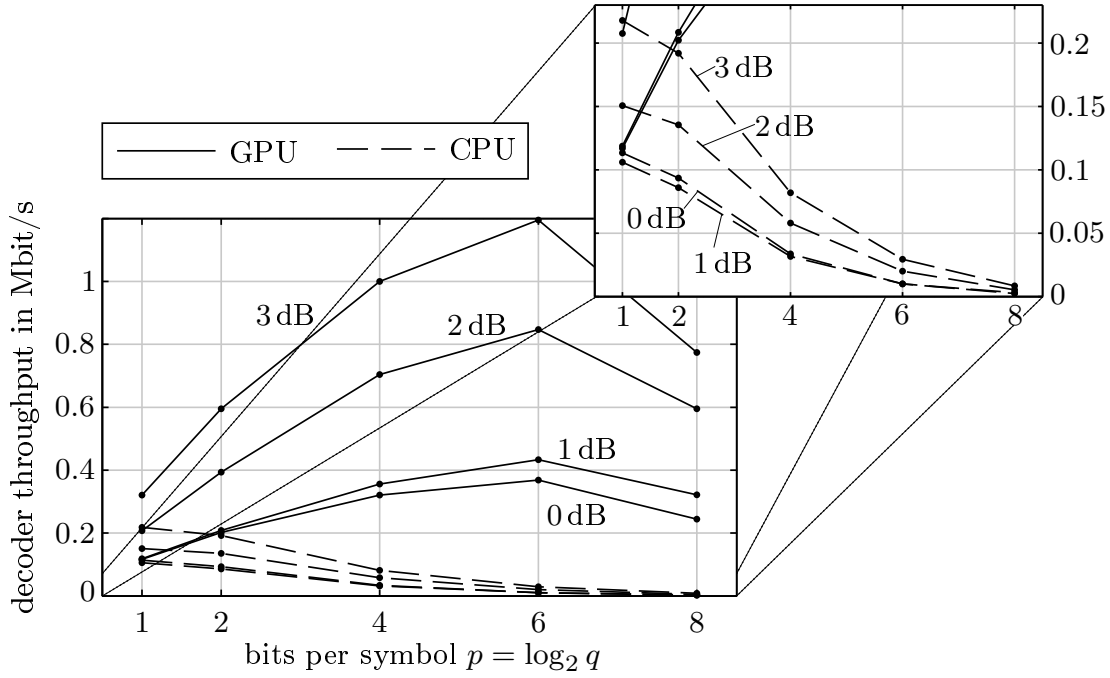


Figure 6.12: Throughput over Galois field dimension of the layered schedule for $E_b/N_0|_{\text{dB}} \in \{0 \text{ dB}, 1 \text{ dB}, 2 \text{ dB}, 3 \text{ dB}\}$

Finally, the speedup achieved by the GPU implementation with respect to the CPU implementation is shown in Table 6.3 for several E_b/N_0 values. Clearly, for small Galois field dimensions, the speedup is much smaller than for the flooding implementation. However, the larger the Galois field dimension gets (and thus the number of threads), the closer the speedup values approach the ones achieved by the flooding schedule.

$E_b/N_0 _{\text{dB}}$	$\mathbb{F}_2$	$\mathbb{F}_4$	$\mathbb{F}_{16}$	$\mathbb{F}_{64}$	$\mathbb{F}_{256}$
0 dB	1.03	2.2	9.6	37	109
1 dB	1.12	2.4	11.3	43	110
2 dB	1.38	2.9	12.1	42	111
3 dB	1.47	3.1	12.2	41	92

Table 6.3: Achieved Speedup $\frac{\text{throughput GPU}}{\text{throughput CPU}}$ of layered schedule.

Summary

To facilitate the increasingly diverse requirements of current and future heterogeneous communication systems and the participating devices, new design strategies for efficient *forward error correction* schemes need to be developed. To meet this demand, this thesis presents different contributions that incorporate enhanced flexibility into the design of LDPC codes and their decoding algorithms. They include highly flexible design methods for the optimization and construction of *multi-rate* LDPC codes and *multi-length* LDPC codes as well as combinations of multi-rate and multi-length capabilities. The goal of these methods is to provide a general LDPC coding framework that renders the specification of multiple separate LDPC codes unnecessary. Instead, communication systems are enabled to flexibly adapt the code rate and length using unified encoder and decoder implementations for only a single LDPC *mother code*. To guarantee an efficient use of resources it is necessary to minimize the inherent performance degradation that comes with this restriction to a single LDPC code. Additionally, contributions are presented that can be incorporated into existing systems to increase their flexibility and/or improve their decoding performance.

Design of Multi-Rate LDPC Codes

One main result of this thesis is a novel design method for multi-rate LDPC codes. Information shortening inserts known dummy bit values to some information bit positions before encoding to achieve code rates lower than that of the mother code. By analytically modeling the shortened degree distributions, an optimization method is presented to obtain shortened code ensembles with close-to-capacity error correction performance. Furthermore, two finite length optimization techniques that select specific positions within the information word of a given LDPC code are presented. Their objective is a uniform distribution of perfect knowledge throughout the Tanner graph (Maximum Distance) and the removal of as many short cycles as possible (Breaking Cycles), respectively. A significant performance gain over previous shortening strategies is illustrated by error rate simulations. Furthermore, both approaches are directly applicable to existing LDPC codes and can, thus, be integrated into existing systems with almost no additional complexity.

Achieving higher code rates than that of the mother code, in contrast, is accomplished by parity puncturing, i.e., by simply discarding a number of parity bits before

transmission. Good punctured code ensembles are constructed by an asymptotic optimization method that incorporates an analytical model for the effect of punctured bits at the decoder. In addition, a finite length optimization (Guaranteed Recoverability) enables puncturing of a mother code up to arbitrarily high code rates. This method is capable of sustaining good performance for very high puncturing fractions, which is not possible with most previously presented finite length puncturing schemes.

To support arbitrary code rate ranges between 0 and 1, a combination of shortening and puncturing is proposed. By constraining shortening and puncturing to be applied in a rate-compatible manner, a multi-rate scheme is obtained that can be realized in terms of a single encoder/decoder pair for the mother code without adding complexity. A joint optimization problem is formulated and solved, that results in asymptotically optimal shortening and puncturing rules. The obtained asymptotic ensemble thresholds surpass those of previous approaches that offer the same degree of flexibility. Based on this and the well-known *progressive edge-growth* (PEG) algorithm, the optimized code construction algorithm *multi-rate improved progressive edge-growth* (MuRiPEG) is proposed. It includes finite length considerations in terms of avoiding short cycles, improving the *approximate cycle extrinsic message degree* (ACE) properties, achieving concentrated check node degree distributions, and reducing the impairment caused by punctured bits. In extensive bit error rate (BER) simulations, MuRiPEG outperforms several previous approaches known from literature and simultaneously offers an increased flexibility in terms of supported code rates. For the exemplarily presented and challenging code rate range from 0.1 to 0.9, a code length of 1008 bits, and a single mother code of rate 0.7, MuRiPEG is capable to achieve almost the same average performance as using one dedicated LDPC code per code rate. An average performance penalty of only 0.06 dB is demonstrated.

The presented advances in the design of multi-rate LDPC codes allow to reduce the hardware complexity as compared to currently specified systems to a large extent. Multiple separately optimized codes can be replaced by one properly designed multi-rate LDPC code. The same flexibility in terms of supported code rates is achieved with an average performance loss of only fractions of a decibel. Additionally, more specifically optimized hardware implementations of encoder and decoder can be employed, since they can be tailored towards a single LDPC code.

Design of Multi-Length LDPC Codes

A further key achievement is the optimization and construction of multi-length LDPC codes. Concurrently shortening and puncturing a mother code word is introduced as an effective method of very low-complexity which is capable of flexibly reducing the code length of an arbitrary LDPC mother code. No other method to adapt the code length has been presented in the literature that provides the same flexibility and is generally applicable to LDPC codes. The proposed method shortens and punctures equal fractions of bits, resulting in a reduced code length while maintaining the same code rate. After introducing asymptotic analysis and optimization, a matrix construction method termed *multi-length improved progressive edge-growth* (MuLiPEG) is proposed in analogy to the MuRiPEG algorithm. Exemplary ensemble threshold

results as well as extensive BER simulations confirm that the range of code lengths for which MuLiPEG offers good error correction performance depends on the target code rate. While for code rates of 0.4 and above, good performance is demonstrated for a reduction of the relative code length down to 6.25 %, the performance penalty quickly increases for smaller code rates. This is caused by the large number of bits that need to be punctured in these cases. The conceptional similarity to the multi-rate scheme MuRiPEG opens up the possibility to combine it with MuLiPEG to realize a single LDPC code with multi-rate and multi-length capabilities – a promising approach for further future studies.

The second presented multi-length approach offers even more flexibility regarding the supported code lengths but is restricted to the case of quasi-cyclic (QC) LDPC codes, which are, however, widely used in practical systems. Assuming a QC encoder and decoder architecture that is based only on the *lifting matrix*, a multi-length QC LDPC code can be realized by *expanding* this lifting matrix with several *lifting factors*. The novel method *multi-length quasi-cyclic progressive edge-growth* (MuLQPEG) constructs a lifting matrix by jointly considering the performance of all expanded check matrices. This method yields a multi-length QC LDPC code defined by a single lifting matrix that supports several code lengths. BER simulations for the range of code lengths from 140 to 32760 show that MuLQPEG does not cause a noticeable performance degradation compared to the case of specifically optimized codes at all code lengths. This remarkable achievement distinguishes MuLQPEG from two comparable approaches from literature and also from MuLiPEG, since all of these exhibit an explicit performance loss. As a resulting design guideline for multi-length setups, QC LDPC codes together with MuLQPEG should be the first choice to enable extensive flexibility.

Combination of Multi-Rate and Multi-Length LDPC Codes

Finally, the construction of a highly flexible multi-rate-and-multi-length QC LDPC code by combining MuRiPEG and MuLQPEG concludes the presented code design methodology. No other method is currently known that can simultaneously support arbitrary ranges of code rates and code lengths with only a single mother code. While being the first method to actually offer this considerable flexibility, in addition, the presented algorithm closely approaches the performance of dedicated LDPC codes for each combination of code rate and length. In an exemplary simulation setup with code rates between 0.1 and 0.9 and code lengths between 210 and 32760, a single proposed code is compared to 63 dedicated LDPC codes. If averaging over all code rate and length combinations, the total penalty of replacing these 63 codes by a single code is only 0.076 dB and thus practically negligible.

Altogether, the contributions presented in the fields of multi-rate and multi-length LDPC code design establish a novel degree of flexibility. While most current systems specify multiple LDPC codes, wide ranges of code rates and code lengths can now be covered by a single mother code. With almost no penalty in error correction performance, the complexity of encoding and decoding can be significantly reduced to

that of only the common mother LDPC code. As a result, the provided LDPC coding framework now meets the requirements of future mobile communication systems.

Advanced Decoding Algorithms and Implementation Aspects

Besides incorporating more flexibility into LDPC code design, also advances in the flexibility of LDPC decoding algorithms are achieved. Two novel classes of decoder schedules for *belief propagation* (BP) decoding are introduced, that are suitable for different types of devices by meeting different requirements in terms of residual error rates, computational complexity, and convergence speed.

The proposed *probabilistic sequential* decoder schedules combine the idea of successively updating single nodes within each iteration with a *girth* dependent updating probability. For both binary and non-binary LDPC codes, they are shown to exhibit similarly fast convergence as previous sequential schedules. At the same time they achieve lower residual error rates if a large number of iterations is conducted. Since the additionally induced complexity is negligible, these schedules are equally suited for low-complexity devices that can only execute a small number of iterations and for delay-insensitive applications that target lowest possible error rates.

Moreover, two novel *informed dynamic scheduling* (IDS) strategies termed (*lazy*) *queue residual decoding* ((L)QRD) are proposed. They improve the concept of previous IDS methods by maintaining an adaptive list of messages that have a large impact on decoding convergence, i.e., that have *residuals* greater than a flexibly adjustable threshold. Thereby, the complexity of searching for messages to be updated is reduced by multiple orders of magnitude. At the same time, lower error rates and in some cases even faster convergence speed is achieved compared to previous IDS and also non-dynamic strategies. Consequently, the proposed (L)QRD schedules are suitable for high-end devices that offer high compute capabilities and target extremely fast convergence (e.g., for low-delay purposes) and/or best possible error correction performance.

Finally, practical implementation aspects of non-binary BP decoding are discussed and a generic implementation on a *graphics processing unit* (GPU) is presented. Non-binary LDPC codes over higher order Galois fields are of special interest, as they outperform binary LDPC codes especially for moderate code lengths and are, thus, attractive candidates to improve the error correction performance of low-delay applications. The presented massively parallelized decoder architecture is the first design that is generally applicable to arbitrary LDPC codes without any algorithmic approximations of the BP algorithm. It achieves up to 148-fold speedup in the presented simulation examples. As alternative to the conventional flooding schedule, a parallelized architecture for the *layered* decoding of non-binary QC LDPC codes with up to 110-fold speedup is presented, which is of particular practical relevance due to many advantages in terms of hardware aspects of both the layered schedule and also QC LDPC codes.

A

Derivation of Maximum Average Relative Code Length

In Section 3.3 on pages 79 ff., the scenario of covering an arbitrary code rate range $r_{\min}$ to $r_{\max}$ with a single mother code of rate r (with $r_{\min} \leq r \leq r_{\max}$) using information shortening to achieve code rates $r_s < r$ and parity puncturing to achieve code rates $r_p > r$ is considered. In this context, it is stated that the maximum average relative code length is given by

$$\bar{n}_{\text{eff}}^* \stackrel{(3.40)}{=} \frac{\log(1 + r_{\max} - r_{\min})}{r_{\max} - r_{\min}}$$

and that it is attained at the mother code rate

$$r^* \stackrel{(3.41)}{=} \frac{1}{1 + \frac{1-r_{\min}}{r_{\max}}}.$$

A proof of this statement is given in the following.

Under the assumption of equiprobable and densely spaced target code rates between $r_{\min}$ and $r_{\max}$, and with $\mathbf{E}\{\cdot\}$ denoting the expectation operator, the average relative code length $\bar{n}_{\text{eff}}$ is given by

$$\begin{aligned} \bar{n}_{\text{eff}} &= \mathbf{P}(\text{“shortening is used”}) \mathbf{E}\{n_s\} + \mathbf{P}(\text{“puncturing is used”}) \mathbf{E}\{n_p\} \\ &= \frac{r - r_{\min}}{r_{\max} - r_{\min}} \mathbf{E}\{n_s\} + \frac{r_{\max} - r}{r_{\max} - r_{\min}} \mathbf{E}\{n_p\} \\ &= \frac{r - r_{\min}}{r_{\max} - r_{\min}} \int_{r_{\min}}^r n_s \mathbf{p}(r_s) \, dr_s + \frac{r_{\max} - r}{r_{\max} - r_{\min}} \int_r^{r_{\max}} n_p \mathbf{p}(r_p) \, dr_p \end{aligned}$$

$$\begin{aligned}
&= \frac{r - r_{\min}}{r_{\max} - r_{\min}} \int_{r_{\min}}^r \frac{n_s}{r - r_{\min}} dr_s + \frac{r_{\max} - r}{r_{\max} - r_{\min}} \int_r^{r_{\max}} \frac{n_p}{r_{\max} - r} dr_p \\
&= \frac{1}{r_{\max} - r_{\min}} \int_{r_{\min}}^r n_s dr_s + \frac{1}{r_{\max} - r_{\min}} \int_r^{r_{\max}} n_p dr_p \\
&= \frac{1}{r_{\max} - r_{\min}} \left(\int_{r_{\min}}^r \frac{1 - r}{1 - r_s} dr_s + \int_r^{r_{\max}} \frac{r}{r_p} dr_p \right) \\
&= \frac{1}{r_{\max} - r_{\min}} \left((1 - r) [-\log(1 - r_s)]_{r_{\min}}^r + r [\log(r_p)]_r^{r_{\max}} \right) \\
&= \frac{1}{r_{\max} - r_{\min}} \left((1 - r) \log \frac{1 - r_{\min}}{1 - r} + r \log \frac{r_{\max}}{r} \right). \tag{A.1}
\end{aligned}$$

The first derivative of $\bar{n}_{\text{eff}}$ with respect to r is computed by:

$$\begin{aligned}
\frac{d\bar{n}_{\text{eff}}}{dr} &= \frac{d}{dr} \left(\frac{1}{r_{\max} - r_{\min}} \left((1 - r) \log \frac{1 - r_{\min}}{1 - r} + r \log \frac{r_{\max}}{r} \right) \right) \\
&= \frac{d}{dr} \left((1 - r) \log(1 - r_{\min}) - (1 - r) \log(1 - r) + r \log(r_{\max}) - r \log(r) \right) \\
&= -\log(1 - r_{\min}) + 1 + \log(1 - r) + \log(r_{\max}) - 1 - \log(r) \\
&= \log \frac{1 - r}{r} - \log \frac{1 - r_{\min}}{r_{\max}}. \tag{A.2}
\end{aligned}$$

Setting the first derivative of $\bar{n}_{\text{eff}}$ to zero yields the optimal mother code rate r^* :

$$\begin{aligned}
&\frac{d\bar{n}_{\text{eff}}}{dr} \stackrel{!}{=} 0 \\
\Leftrightarrow \quad \log \frac{1 - r}{r} - \log \frac{1 - r_{\min}}{r_{\max}} &= 0 \\
\Leftrightarrow \quad \frac{1 - r}{r} &= \frac{1 - r_{\min}}{r_{\max}} \\
\Leftrightarrow \quad (1 - r)r_{\max} &= (1 - r_{\min})r \\
\Leftrightarrow \quad r^* &= \frac{r_{\max}}{r_{\max} + 1 - r_{\min}} \\
\Leftrightarrow \quad r^* &= \frac{1}{1 + \frac{1 - r_{\min}}{r_{\max}}}. \tag{A.3}
\end{aligned}$$

Since the second derivative

$$\begin{aligned}
\frac{d^2\bar{n}_{\text{eff}}}{dr^2} &= \frac{d}{dr} \left(\log \frac{1 - r}{r} - \log \frac{1 - r_{\min}}{r_{\max}} \right) \\
&= \frac{-1}{(r_{\max} - r_{\min})(1 - r)r} < 0, \quad \text{for } 0 < r < 1 \tag{A.4}
\end{aligned}$$

is always less than zero, the extreme value is indeed a maximum. Inserting the optimal mother code rate $r^\star$ from (A.3) into $\bar{n}_{\text{eff}}$ from (A.1) yields the maximum relative code length $\bar{n}_{\text{eff}}^\star$ as

$$\begin{aligned}\bar{n}_{\text{eff}}^\star &= \frac{1}{r_{\text{max}} - r_{\text{min}}} \left((1 - r^\star) \log \frac{1 - r_{\text{min}}}{1 - r^\star} + r^\star \log \frac{r_{\text{max}}}{r^\star} \right) \\ &= \frac{\log(1 + r_{\text{max}} - r_{\text{min}})}{r_{\text{max}} - r_{\text{min}}}.\end{aligned}\tag{A.5}$$

Bibliography

- [3rd08] 3rd Generation Partnership Project (3GPP); Technical Specification Group Radio Access Network. “Evolved Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding”. TS 36.212, Version 8.2.0, Rel. 8, March 2008.
- [3rd10] 3rd Generation Partnership Project (3GPP); Technical Specification Group Radio Access Network. “Evolved Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation”. TS 36.211, Version 9.0.0, Rel. 9, January 2010.
- [AB13] R. Asvadi and A. Banihashemi. “A Rate-Compatible Puncturing Scheme for Finite-Length LDPC Codes”. *IEEE Communications Letters*, vol. 17, no. 1, pp. 147–150, January 2013.
- [AFSK13] J. Andrade, G. Falcao, V. Silva, and K. Kasai. “FFT-SPA Non-Binary LDPC Decoding on GPU”. *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.
- [BBV11] M. Beermann, T. Breddermann, and P. Vary. “Rate-Compatible LDPC Codes using Optimized Dummy Bit Insertion”. *Proceedings IEEE International Symposium on Wireless Communication Systems (ISWCS)*, November 2011.
- [BD03] L. Barnault and D. Declercq. “Fast Decoding Algorithm for LDPC over $GF(2^q)$ ”. *Proceedings IEEE Information Theory Workshop (ITW)*, pp. 70–73, March 2003.
- [Bee10] M. Beermann. “Entwurf und Optimierung von LDPC- und Turbo-Codes für Hochgeschwindigkeits-Mobilfunknetze”. Diplomarbeit, IND, RWTH Aachen, May 2010.
- [BEV11] T. Breddermann, B. Eschbach, and P. Vary. “Dummy Bit Rate Matching for UMTS LTE”. *ITG Fachtagung Mobilkommunikation*, Osnabrück, Germany, May 2011.
- [BG96] C. Berrou and A. Glavieux. “Near Optimum Error Correcting Coding and Decoding: Turbo-Codes”. *IEEE Transactions Communications*, vol. 44, no. 10, pp. 1261–1271, October 1996.
- [BGT93] C. Berrou, A. Glavieux, and P. Thitimajshima. “Near Shannon Limit Error-Correcting Coding and Decoding”. *Proceedings IEEE International Conference on Communications (ICC)*, pp. 1064–1070, Geneva, Switzerland, 1993.
- [BMSV13] M. Beermann, E. Monzó, L. Schmalen, and P. Vary. “High Speed Decoding of Non-Binary Irregular LDPC Codes Using GPUs”. *Proceedings IEEE Workshop on Signal Processing Systems (SiPS)*, October 2013. Awarded as “Best Paper”.

- [BMSV15] M. Beermann, E. Monzó, L. Schmalen, and P. Vary. “GPU Accelerated Belief Propagation Decoding of Non-Binary LDPC Codes with Parallel and Sequential Scheduling”. *Springer Journal of Signal Processing Systems*, vol. 78, no. 1, pp. 21–34, January 2015.
- [BP06] D. Bi and L. Perez. “Rate-Compatible Low-Density Parity-Check Codes with Rate-Compatible Degree Profiles”. *Electronics Letters*, vol. 42, no. 1, January 2006.
- [BPL13] A. Bazarsky, N. Presman, and S. Litsyn. “Design of Non-Binary Quasi-Cyclic LDPC Codes by ACE Optimization”. *Proceedings IEEE Information Theory Workshop (ITW)*, pp. 1–5, September 2013.
- [Bre14] T. Breddermann. *On Rate-Compatible Insertion Convolutional Turbo Codes and HARQ for Mobile Communications*. Dissertation, IND, RWTH Aachen, January 2014.
- [BSV11] M. Beermann, L. Schmalen, and P. Vary. “Improved Decoding of Binary and Non-Binary LDPC Codes by Probabilistic Shuffled Belief Propagation”. *Proceedings IEEE International Conference on Communications (ICC)*, June 2011.
- [BT04] G. J. Byers and F. Takawira. “Fourier Transform Decoding of Non-Binary LDPC Codes”. *Proceedings Southern African Telecommunication Networks and Applications Conference (SATNAC)*, September 2004.
- [BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.
- [BV11] M. Beermann and P. Vary. “A Generalization of Residual Belief Propagation for Flexible Reduced Complexity LDPC Decoding”. *Proceedings IEEE Vehicular Technology Conference (VTC Fall)*, September 2011.
- [BV13] M. Beermann and P. Vary. “Breaking Cycles with Dummy Bits: Improved Rate-Compatible LDPC Codes with Short Block Lengths”. *Proceedings International ITG Conference on Systems, Communications, and Coding (SCC)*, January 2013.
- [BV14] M. Beermann and P. Vary. “Joint Optimization of Multi-Rate LDPC Code Ensembles for the AWGN Channel Based on Shortening and Puncturing”. *Proceedings IEEE Wireless Communications and Networking Conference (WCNC)*, April 2014.
- [BWV14] M. Beermann, F. Wickert, and P. Vary. “Highly Flexible Design of Multi-Rate Multi-Length Quasi-Cyclic LDPC Codes”. *Proceedings International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*. IEEE, September 2014.
- [CC09] T. P. Chen and Y.-K. Chen. “Challenges and Opportunities of Obtaining Performance from Multi-Core CPUs and Many-Core GPUs”. *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2009.
- [CCG79] J. Cain, G. Clark, and J. Geist. “Punctured Convolutional Codes of Rate $(n-1)/n$ and Simplified Maximum Likelihood Decoding”. *IEEE Transactions Information Theory*, vol. 25, no. 1, pp. 97–100, January 1979.

- [CFH14] W. H. Chin, Z. Fan, and R. Haines. “Emerging Technologies and Research Challenges for 5G Wireless Networks”. *IEEE Wireless Communications*, vol. 21, no. 2, pp. 106–112, April 2014.
- [CFRU01] S. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke. “On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit”. *IEEE Communications Letters*, vol. 5, no. 1, pp. 58–60, February 2001.
- [CGW07a] A. I. V. Casado, M. Griot, and R. Wesel. “Informed Dynamic Scheduling for Belief-Propagation Decoding of LDPC Codes”. *Proceedings IEEE International Conference on Communications (ICC)*, Glasgow, Scotland, June 2007.
- [CGW07b] A. I. V. Casado, M. Griot, and R. Wesel. “Improving LDPC Decoders via Informed Dynamic Scheduling”. *Proceedings IEEE Information Theory Workshop (ITW)*, Lake Tahoe, CA, USA, September 2007.
- [CGW10] A. Casado, M. Griot, and R. Wesel. “LDPC Decoders with Informed Dynamic Scheduling”. *IEEE Transactions Communications*, vol. 58, no. 12, pp. 3470–3479, December 2010.
- [CH93] O. Collins and M. Hizlan. “Determinate State Convolutional Codes”. *IEEE Transactions Communications*, vol. 41, no. 12, pp. 1785–1794, December 1993.
- [Coo12] S. Cook. *CUDA Programming: A Developer’s Guide to Parallel Computing With GPUs*. Morgan Kaufmann, 2012.
- [CRU01] S. Chung, T. J. Richardson, and R. Urbanke. “Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation”. *IEEE Transactions Information Theory*, vol. 47, no. 2, pp. 657–670, February 2001.
- [DM98a] M. C. Davey and D. J. C. MacKay. “Low-Density Parity Check Codes over $GF(q)$ ”. *IEEE Communications Letters*, vol. 2, no. 6, pp. 165–167, June 1998.
- [DM98b] M. C. Davey and D. J. C. MacKay. “Low-Density Parity Check Codes over $GF(q)$ ”. *Proceedings IEEE Information Theory Workshop (ITW)*, June 1998.
- [EMK06] G. Elidan, I. McGraw, and D. Koller. “Residual Belief Propagation: Informed Scheduling for Asynchronous Message Passing”. *Proceedings Conference on Uncertainty in Artificial Intelligence*, Cambridge, MA, USA, July 2006.
- [Eri13] Ericsson. “5G Radio Access - Research and Vision”, June 2013. White Papers, http://www.ericsson.com/news?categoryFilter=white_papers.
- [ETV99] T. Etzion, A. Trachtenberg, and A. Vardy. “Which Codes Have Cycle-Free Tanner Graphs?”. *IEEE Transactions Information Theory*, vol. 45, no. 6, pp. 2173–2181, September 1999.
- [Eur00] European Telecommunications Standards Institute (ETSI). “EN 301 704: Digital Cellular Telecommunications System (phase 2+) (GSM); Adaptive Multi-Rate (AMR) Speech Transcoding (GSM 06.90 version 7.2.1 release 1998)”, 2000.
- [Eur09a] European Telecommunications Standards Institute (ETSI). “EN 302307: Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)”, August 2009.

- [Eur09b] European Telecommunications Standards Institute (ETSI). “EN 302755: Digital Video Broadcasting (DVB); Frame structure, channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2)”, September 2009.
- [FMI99] M. Fossorier, M. Mihaljevic, and H. Imai. “Reduced Complexity Iterative Decoding of Low-Density Parity Check Codes Based on Belief Propagation”. *IEEE Transactions Communications*, vol. 47, no. 5, pp. 673–680, May 1999.
- [Gal62] R. G. Gallager. “Low-Density Parity-Check Codes”. *IRE Transactions Information Theory*, pp. 21–28, 1962.
- [Gal63] R. G. Gallager. *Low-Density Parity-Check Codes*. M.I.T. Press, Cambridge, MA, USA, 1963.
- [GSD10] M. Gorgoglione, V. Savin, and D. Declercq. “Optimized Puncturing Distributions for Irregular Non-Binary LDPC Codes”. *Proceedings International Symposium on Information Theory and its Applications (ISITA)*, pp. 400–405, October 2010.
- [Hag88] J. Hagenauer. “Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and their Applications”. *IEEE Transactions Communications*, vol. 36, no. 4, pp. 389–400, April 1988.
- [Hal08] T. R. Halfhill. “Parallel Processing with CUDA”. *Microprocessor Report*, January 2008.
- [HC06] T. Halford and K. Chugg. “An Algorithm for Counting Short Cycles in Bipartite Graphs”. *IEEE Transactions Information Theory*, vol. 52, no. 1, pp. 287–292, January 2006.
- [HEA01] X. Y. Hu, E. Eleftheriou, and D. M. Arnold. “Progressive Edge-Growth Tanner Graphs”. *Proceedings IEEE Global Telecommunications Conference (GLOBECOM)*, vol. 2, pp. 995–1001, November 2001.
- [HEA05] X. Y. Hu, E. Eleftheriou, and D. M. Arnold. “Regular and Irregular Progressive Edge-Growth Tanner Graphs”. *IEEE Transactions Information Theory*, vol. 51, no. 1, pp. 386–398, January 2005.
- [HKKM06] J. Ha, J. Kim, D. Klinc, and S. McLaughlin. “Rate-Compatible Punctured Low-Density Parity-Check Codes with Short Block Lengths”. *IEEE Transactions Information Theory*, vol. 52, no. 2, pp. 728–738, February 2006.
- [HKM04] J. Ha, J. Kim, and S. McLaughlin. “Rate-Compatible Puncturing of Low-Density Parity-Check Codes”. *IEEE Transactions Information Theory*, vol. 50, no. 11, pp. 2824–2836, November 2004.
- [HLZZ10] J. Huang, L. Liu, W. Zhou, and S. Zhou. “Large-Girth Nonbinary QC-LDPC Codes of Various Lengths”. *IEEE Transactions Communications*, vol. 58, no. 12, pp. 3436–3447, December 2010.
- [HM03a] J. Ha and S. McLaughlin. “Optimal Puncturing of Irregular Low-Density Parity-Check Codes”. *Proceedings IEEE International Conference on Communications (ICC)*, vol. 5, pp. 3110–3114, May 2003.
- [HM03b] J. Ha and S. McLaughlin. “Optimal Puncturing Distributions for Rate-Compatible Low-Density Parity-Check Codes”. *Proceedings IEEE International Symposium on Information Theory (ISIT)*, pp. 233–233, June 2003.

- [Hoc04] D. Hocevar. “A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes”. *Proceedings IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 107–112, 2004.
- [HOP96] J. Hagenauer, E. Offer, and L. Papke. “Iterative Decoding of Binary Block and Convolutional Codes”. *IEEE Transactions Communications*, vol. 42, no. 2, pp. 429–445, March 1996.
- [IEE09a] IEEE Computer Society. “IEEE Standard for Local and metropolitan area networks; Part 16: Air Interface for Broadband Wireless Access Systems”. IEEE Std. 802.16-2009, June 2009.
- [IEE09b] IEEE Standards Association. “Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput”. IEEE Std 802.11n-2009, 2009.
- [IEE10] IEEE Standards Association. “Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications”. IEEE Std 1901-2010, December 2010.
- [JDA⁺07] C. Jones, S. Dolinar, K. Andrews, D. Divsalar, Y. Zhang, and W. Ryan. “Functions and Architectures for LDPC Decoding”. *Proceedings IEEE Information Theory Workshop (ITW)*, pp. 577–583, September 2007.
- [JFZ99] A. Jimenez Felstrom and K. Zigangirov. “Time-Varying Periodic Convolutional Codes with Low-Density Parity-Check Matrix”. *IEEE Transactions Information Theory*, vol. 45, no. 6, pp. 2181–2191, September 1999.
- [JKM00] H. Jin, A. Khandekar, and R. McEliece. “Irregular Repeat-Accumulate Codes”. *Proceedings International Symposium on Turbo Codes and Related Topics*, pp. 1–8, September 2000.
- [KB10] M. Karimi and A. Banihashemi. “A Message-Passing Algorithm for Counting Short Cycles in a Graph”. *Proceedings IEEE Information Theory Workshop (ITW)*, January 2010.
- [KB13] M. Karimi and A. Banihashemi. “Message-Passing Algorithms for Counting Short Cycles in a Graph”. *IEEE Transactions Communications*, vol. 61, no. 2, pp. 485–495, February 2013.
- [KCY12] K.-J. Kim, J.-H. Chung, and K. Yang. “Design of Length-Compatible Low-Density Parity-Check Codes”. *IEEE Communications Letters*, vol. 16, no. 5, pp. 734–737, May 2012.
- [KDPS10] K. Kasai, D. Declercq, C. Poulliat, and K. Sakaniwa. “Rate-Compatible Non-Binary LDPC Codes Concatenated with Multiplicative Repetition Codes”. *Proceedings IEEE International Symposium on Information Theory (ISIT)*, pp. 844–848, June 2010.
- [KDPS11] K. Kasai, D. Declercq, C. Poulliat, and K. Sakaniwa. “Multiplicatively Repeated Nonbinary LDPC Codes”. *IEEE Transactions Information Theory*, vol. 57, no. 10, pp. 6788–6795, October 2011.
- [KGJV83] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi. “Optimization by Simulated Annealing”. *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
- [KGP06] D. Kimura, F. Guilloud, and R. Pyndiah. “Construction of Parity-Check Matrices for Non-Binary LDPC Codes”. *Proceedings International Symposium on Turbo Codes and Related Topics*, April 2006.

- [KHM08] D. Kline, J. Ha, and S. McLaughlin. “On Rate-Adaptability of Nonbinary LDPC Codes”. *Proceedings International Symposium on Turbo Codes and Related Topics*, pp. 231–236, September 2008.
- [Khr] Khronos Group. “OpenCL – the open standard for parallel programming of heterogeneous systems”. <http://khronos.org/opencl/> [accessed Jan. 2015].
- [KK03] H. Kfir and I. Kanter. “Parallel Versus Sequential Updating for Belief Propagation Decoding”. *Physica A: Statistical Mechanics and its Applications*, vol. 330, no. 1-2, pp. 259–270, December 2003. Randomness and Complexity: Proceedings of the International Workshop in Honor of Shlomo Havlin’s 60th birthday.
- [KLF01] Y. Kou, S. Lin, and M. Fossorier. “Low-Density Parity-Check Codes Based on Finite Geometries: a Rediscovery and New Results”. *IEEE Transactions Information Theory*, vol. 47, no. 7, pp. 2711–2736, November 2001.
- [LCZ⁺06] Z. Li, L. Chen, L. Zeng, S. Lin, and W. Fong. “Efficient Encoding of Quasi-Cyclic Low-Density Parity-Check Codes”. *IEEE Transactions Communications*, vol. 54, no. 1, pp. 71–81, January 2006.
- [LFK09] G. Li, I. Fair, and W. Krzymien. “Density Evolution for Nonbinary LDPC Codes Under Gaussian Approximation”. *IEEE Transactions Information Theory*, vol. 55, no. 3, pp. 997–1015, March 2009.
- [LK04] Z. Li and B. V. Kumar. “A Class of Good Quasi-Cyclic Low-Density Parity Check Codes Based on Progressive Edge Growth Graph”. *Proceedings Asilomar Conference on Signals, Systems, and Computers*, November 2004.
- [LMSS98a] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman. “Analysis of Low Density Codes and Improved Designs Using Irregular Graphs”. *Proceedings 30th Annual ACM Symposium on Theory of Computing (STOC)*, May 1998.
- [LMSS98b] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman. “Improved Low-Density Parity-Check Codes Using Irregular Graphs and Belief Propagation”. *Proceedings IEEE International Symposium on Information Theory (ISIT)*, August 1998.
- [LMSS01] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman. “Improved Low-Density Parity-Check Codes Using Irregular Graphs”. *IEEE Transactions Information Theory*, vol. 47, no. 2, pp. 585–598, February 2001.
- [LN02] J. Li and K. Narayanan. “Rate-Compatible Low Density Parity Check Codes for Capacity-Approaching ARQ Schemes in Packet Data Communications”. *Proceedings International Conference on Communications, Internet, and Information Technology*, pp. 201–206, Virgin Islands, USA, November 2002.
- [LS13] A. Leven and L. Schmalen. *Optical Fiber Telecommunications Volume VI-A*, chapter 15: Implementation Aspects of Coherent Transmit and Receive Functions in Application-Specific Integrated Circuits. Academic Press, June 2013.
- [LSL07] D. Levin, E. Sharon, and S. Litsyn. “Lazy Scheduling for LDPC Decoding”. *IEEE Communications Letters*, vol. 11, no. 1, pp. 70–72, January 2007.
- [LWZ09] X. Liu, X. Wu, and C. Zhao. “Shortening for Irregular QC-LDPC Codes”. *IEEE Communications Letters*, vol. 13, no. 8, pp. 612–614, August 2009.

- [Mac99] D. J. C. MacKay. “Good Error-Correcting Codes Based on Very Sparse Matrices”. *IEEE Transactions Information Theory*, vol. 45, no. 2, pp. 399–431, March 1999.
- [MB01a] Y. Mao and A. H. Banihashemi. “A New Schedule for Decoding Low-Density Parity-Check Codes”. *Proceedings IEEE Global Telecommunications Conference (GLOBECOM)*, vol. 2, pp. 1007–1010, 2001.
- [MB01b] Y. Mao and A. H. Banihashemi. “Decoding Low-Density Parity-Check Codes With Probabilistic Scheduling”. *IEEE Communications Letters*, vol. 5, no. 10, pp. 414–416, October 2001.
- [MN95] D. J. C. MacKay and R. M. Neal. “Good Codes Based on Very Sparse Matrices”. *5th IMA Conference on Cryptography and Coding*, pp. 100–111, Springer, Berlin, Germany, 1995.
- [Mon10] E. Monzó. “Massive Parallel Decoding of Low-Density Parity-Check Codes Using Graphic Cards”. Master’s thesis, RWTH Aachen University & Universidad Politecnica de Valencia, 2010. available online: <http://riunet.upv.es/bitstream/handle/10251/13736/Thesis%20-%20Enrique%20Monz%C3%B3%20Solves-1.pdf> [accessed Jan. 2015].
- [Moo05] T. K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, June 2005.
- [MS03] M. M. Mansour and N. R. Shanbhag. “High Throughput LDPC Decoders”. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 976–996, December 2003.
- [MY05] S. Myung and K. Yang. “Extension of Quasi-Cyclic LDPC Codes by Lifting”. *Proceedings IEEE International Symposium on Information Theory (ISIT)*, pp. 2305–2309, September 2005.
- [MYK05] S. Myung, K. Yang, and J. Kim. “Quasi-Cyclic LDPC Codes for Fast Encoding”. *IEEE Transactions Information Theory*, vol. 51, no. 8, pp. 2894–2901, August 2005.
- [NH09] C.-I. C. Nilsen and I. Hafizovic. “Digital Beamforming Using a GPU”. *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2009.
- [Oka08] T. Okamura. “A Hybrid ARQ Scheme Based on Shortened Low-Density Parity-Check Codes”. *Proceedings IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 82–87, April 2008.
- [PC07] R.-H. Peng and R.-R. Chen. “Design of Nonbinary Quasi-Cyclic LDPC Cycle Codes”. *Proceedings IEEE Information Theory Workshop (ITW)*, pp. 13–18, September 2007.
- [PDN14] S. Pfletschinger, D. Declercq, and M. Navarro. “Adaptive HARQ With Non-Binary Repetition Coding”. *IEEE Transactions on Wireless Communications*, vol. 13, no. 8, pp. 4193–4204, August 2014.
- [PFD08] C. Poulliat, M. Fossorier, and D. Declercq. “Design of Regular $(2, d_c)$ -LDPC Codes over $GF(q)$ Using their Binary Images”. *IEEE Transactions Communications*, vol. 56, no. 10, pp. 1626–1635, October 2008.
- [PNF07] H. Pishro-Nik and F. Fekri. “Results on Punctured Low-Density Parity-Check Codes and Improved Iterative Decoding Techniques”. *IEEE Transactions Information Theory*, vol. 53, no. 2, pp. 599–614, February 2007.

- [RC12] D. Romero and N. Chang. “Sequential Decoding of Non-Binary LDPC Codes on Graphics Processing Units”. *Proceedings Asilomar Conference on Signals, Systems, and Computers*, November 2012.
- [Ric03] T. Richardson. “Error Floors of LDPC Codes”. *Proceedings Allerton Conference on Communication, Control, and Computing*, Monticello, IL, USA, October 2003.
- [RL09] W. Ryan and S. Lin. *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.
- [RN03] T. Richardson and V. Novichkov. “Methods and Apparatus for Decoding LDPC Codes”, October 2003. US Patent 6,633,856.
- [RSU01] T. Richardson, M. Shokrollahi, and R. Urbanke. “Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes”. *IEEE Transactions Information Theory*, vol. 47, no. 2, pp. 619–637, February 2001.
- [RU01a] T. Richardson and R. Urbanke. “Efficient Encoding of Low-Density Parity-Check Codes”. *IEEE Transactions Information Theory*, vol. 47, no. 2, pp. 638–656, February 2001.
- [RU01b] T. Richardson and R. Urbanke. “The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding”. *IEEE Transactions Information Theory*, vol. 47, no. 2, pp. 599–618, February 2001.
- [RU05] V. Rathi and R. Urbanke. “Density Evolution, Thresholds and the Stability Condition for Non-Binary LDPC Codes”. *IEE Proceedings Communications*, vol. 152, no. 6, pp. 1069–1074, December 2005.
- [RU08] T. Richardson and R. Urbanke. *Modern Coding Theory*. Cambridge University Press, March 2008.
- [SAL06] E. Sharon, A. Ashikhmin, and S. Litsyn. “Analysis of Low-Density Parity-Check Codes Based on EXIT Functions”. *IEEE Transactions Communications*, vol. 54, no. 8, pp. 1407–1414, August 2006.
- [SC03] H. Song and J. Cruz. “Reduced-Complexity Decoding of Q -ary LDPC Codes for Magnetic Recording”. *IEEE Transactions on Magnetics*, vol. 39, no. 2, pp. 1081–1087, March 2003.
- [Sha48] C. Shannon. “A Mathematical Theory of Communication”. *Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, July and October 1948.
- [SLG04] E. Sharon, S. Litsyn, and J. Goldberger. “An Efficient Message-Passing Schedule for LDPC Decoding”. *Proceedings 23rd IEEE Convention of Electrical and Electronics Engineers in Israel*, September 2004.
- [SLG07] E. Sharon, S. Litsyn, and J. Goldberger. “Efficient Serial Message-Passing Schedules for LDPC Decoding”. *IEEE Transactions Information Theory*, vol. 53, no. 11, pp. 4076–4091, November 2007.
- [SP97] R. Storn and K. Price. “Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”. *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.
- [Tan81] R. M. Tanner. “A Recursive Approach to Low Complexity Codes”. *IEEE Transactions Information Theory*, vol. 27, no. 5, pp. 533–547, September 1981.

- [tB00] S. ten Brink. “Rate One-Half Code for Approaching the Shannon Limit by 0.1 dB”. *Electronics Letters*, vol. 36, no. 15, pp. 1293–1294, July 2000.
- [tB01] S. ten Brink. “Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes”. *IEEE Transactions Communications*, vol. 49, no. 10, pp. 1727–1737, October 2001.
- [tBKA04] S. ten Brink, G. Kramer, and A. Ashikhmin. “Design of Low-Density Parity-Check Codes for Modulation and Detection”. *IEEE Transactions Communications*, vol. 52, no. 4, pp. 670–678, April 2004.
- [TH02] M. Tüchler and J. Hagenauer. “EXIT Charts of Irregular Codes”. *Proceedings Annual Conference on Information Sciences and Systems (CISS)*, pp. 748–753, 2002.
- [TJ05] T. Tian and C. R. Jones. “Construction of Rate-Compatible LDPC Codes Utilizing Information Shortening and Parity Puncturing”. *EURASIP Journal on Wireless Communications and Networking*, vol. 5, pp. 789–795, October 2005.
- [TJVV04] T. Tian, C. Jones, J. Villaseñor, and R. Wesel. “Selective Avoidance of Cycles in Irregular LDPC Code Construction”. *IEEE Transactions Communications*, vol. 52, no. 8, pp. 1242–1247, August 2004.
- [Var97] A. Vardy. “Algorithmic Complexity in Coding Theory and the Minimum Distance Problem”. *Proceedings 29th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 92–109, El Paso, TX, USA, May 1997.
- [VDV⁺10] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard. “Low-Complexity Decoding for Non-Binary LDPC Codes in High Order Fields”. *IEEE Transactions Communications*, vol. 58, no. 5, pp. 1365–1375, May 2010.
- [VF09] B. Vellambi and F. Fekri. “Finite-Length Rate-Compatible LDPC Codes: A Novel Puncturing Scheme”. *IEEE Transactions Communications*, vol. 57, no. 2, pp. 297–301, February 2009.
- [VS09] D. Vukobratovic and V. Senk. “Evaluation and Design of Irregular LDPC Codes Using ACE Spectrum”. *IEEE Transactions Communications*, vol. 57, no. 8, pp. 2272–2279, August 2009.
- [Wic14] F. Wickert. “Entwurf und Analyse von Low-Density-Parity-Check-Codes mit flexibler Blocklänge”. Master’s thesis, RWTH Aachen University, 2014.
- [Win11] W. Winkelhüsener. “Graph-Based Optimization of LDPC Codes”. Student thesis, IND, RWTH Aachen, March 2011.
- [WSM04] H. Wymeersch, H. Steendam, and M. Moeneclaey. “Log-Domain Decoding of LDPC Codes over $GF(q)$ ”. *Proceedings IEEE International Conference on Communications (ICC)*, vol. 2, pp. 772–776, June 2004.
- [WSY⁺12] G. Wang, H. Shen, B. Yin, M. Wu, Y. Sun, and J. R. Cavallaro. “Parallel Non-Binary LDPC Decoding on GPU”. *Proceedings Asilomar Conference on Signals, Systems, and Computers*, November 2012.
- [WZZ⁺11] J. Wu, M. Zhao, J. Zhong, W. Zhang, and X. Lv. “Statistical-Based Density Evolution Algorithm for Nonbinary Low-Density Parity-Check Codes”. *Proceedings IEEE Vehicular Technology Conference (VTC Fall)*, September 2011.

- [XB04] H. Xiao and A. Banihashemi. “Improved Progressive-Edge-Growth (PEG) Construction of Irregular LDPC Codes”. *IEEE Communications Letters*, vol. 8, no. 12, pp. 715–717, December 2004.
- [XLG⁺11] Y. Xu, B. Liu, L. Gong, B. Rong, and L. Gui. “Improved Shortening Algorithm for Irregular QC-LDPC Codes Using Known Bits”. *IEEE Transactions on Consumer Electronics*, vol. 57, no. 3, pp. 1057–1063, August 2011.
- [XR00] W. Xu and J. Romme. “A Class of Multirate Convolutional Codes by Dummy Bit Insertion”. *Proceedings IEEE Global Telecommunications Conference (GLOBECOM)*, San Francisco, CA, USA, November 2000.
- [YB04] M. Yazdani and A. Banihashemi. “On Construction of Rate-Compatible Low-Density Parity-Check Codes”. *IEEE Communications Letters*, vol. 8, no. 3, pp. 159–161, March 2004.
- [YEK⁺07] N. Yacov, H. Efraim, H. Kfir, I. Kanter, and O. Shental. “Parallel vs. Sequential Belief Propagation Decoding of LDPC Codes over GF(q) and Markov Sources”. *Physica A: Statistical Mechanics and its Applications*, vol. 378, no. 2, May 2007.
- [ZBX10] L. Zhou, B. Bai, and M. Xu. “Design of Nonbinary Rate-Compatible LDPC Codes Utilizing Bit-Wise Shortening Method”. *IEEE Communications Letters*, vol. 14, no. 10, pp. 963–965, October 2010.
- [ZF02] J. Zhang and M. Fossorier. “Shuffled Belief Propagation Decoding”. *Proceedings Asilomar Conference on Signals, Systems, and Computers*, vol. 1, pp. 8–15, November 2002.
- [Zha07] Y. Zhang. *Design of Low-Floor Quasi-Cyclic IRA Codes and Their FPGA Decoders*. PhD thesis, Department of Electrical and Computer Engineering, The University of Arizona, May 2007.
- [ZLT⁺08] L. Zeng, L. Lan, Y. Tai, S. Song, S. Lin, and K. Abdel-Ghaffar. “Constructions of Nonbinary Quasi-Cyclic LDPC Codes: A Finite Field Approach”. *IEEE Transactions Communications*, vol. 56, no. 4, pp. 545–554, April 2008.
- [ZMC12] L. Zhang, F. Ma, and L. Cheng. “A Puncturing Scheme for Low-Density Parity-Check Codes Based on 1-SR Nodes”. *Proceedings IEEE Vehicular Technology Conference (VTC Fall)*, September 2012.
- [ZMGC12] H. Zhou, D. Mitchell, N. Goertz, and D. Costello. “A Puncturing Algorithm for Rate-Compatible LDPC Convolutional Codes”. *Proceedings International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pp. 255–259, August 2012.