Chapter 1

Rate-Compatible LDPC and Turbo Codes for Link Adaptivity and Unequal Error Protection

Modern triple-play telecom services (video, audio/voice, data) come with different error protection requirements not only service-related, but also as an inherent property of each service. Scalable video [1,2] and audio codes [3] result in data with different priorities, not just as far as header and data are concerned, but also considering spatial and temporal resolution steps that ask for different protection levels. This ensures that essential information stays undistorted in also somewhat more adverse channel conditions. Such scalable codes have not only been created to realize graceful degradation, but also, since end devices of different qualities (resolutions) ask for inherent provisioning of the right data portion for their display or acoustical devices.

Adaptation of the error correction capability of the deployed codes in these systems is an important problem. Hagenauer’s rate-compatible punctured convolutional codes [4, 5] were the first to provide an elegant solution to this problem. With the advent of capacity-achieving coding schemes, such approaches had to be extended for Turbo, LDPC, and rateless codes. Puncturing and Pruning [6, 7] are still possible choices to adapt rates and error correction performances. However, there are newer techniques such as multi-edge-type LDPC codes and LT codes [8–11] that allow for these capabilities, as well. Our treatment will start from puncturing and pruning in Turbo
Codes in Section 1.1, will then turn to LDPC code methods controlling their degree distributions by puncturing and pruning in Section 1.2, and finally discuss a multi-edge type approach in Section 1.3.

Although not treated in here in detail, it is still worth mentioning that sequential decoders for convolutional codes have a built-in unequal error protection property, decreasing the performance with the decoding depth [12]. Using such properties inside Turbo schemes is not straightforward, due to the reshuffling of data by the interleaver.

Furthermore, there are methods based on bit-loading (adaptive modulation), e.g., [13, 14] or hierarchical modulation [15, 16] that are very flexible in designing SNR gaps between priority classes and could then work with identical codes for all classes or support code designs by shaping channel qualities, i.e., making the channel irregular, although preliminary results regarding mixed bit-loading/LDPC designs are not yet promising [17].

1.1 Unequal Error Protection Turbo Codes

1.1.1 Puncturing and pruning

Puncturing [4, 5] can very easily be described as adapting the rate of convolutional codes, which are the typical constituent codes of a Turbo-coding scheme, by just omitting output bits. This means a rate \( k/n \) code will be transformed into one with rate \( k/(n - p) \), where \( p \) is the number of omitted output bits per output frame of a convolutional code. The receiver is expected to know the position of the punctured bits and in the decoding, will correspondingly set the log-likelihood ratio of these non-transmitted components to zero, i.e., assuming equal probability for \( \pm 1 \) in antipodal signaling.

A puncturing scheme is usually described by means of a puncturing matrix \( P \). This matrix has dimensions \( n \times P \) and consists of ones and zeros. The zeros determine the discarded output symbols (bits). The columns show the time sequence of the puncturing pattern. A column consisting of only ones means that all \( n \) output symbols (bits) of the convolutional encoder will be transmitted.

Let us, e.g., consider a simple example with a convolutional code of rate 1/2 and the generators 5 and 7, i.e., \( 1 + D^2 \) and \( 1 + D + D^2 \), respectively.
A puncturing matrix

\[ P = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \]

will now mean only 3 output bits instead of 4 for 2 input bits. There is also a polynomial description of puncturing, just listing the involved output polynomials.\(^1\) The equivalent trellis of the punctured code is shown in Fig. 1.1, where X stands for the punctured bit.

Puncturing can, of course, be seen as a special case of a more general structure of a code concatenation of the mother code and a code or different codes that have a rate bigger than one. This leads to serial concatenations as illustrated in Fig. 1.2.

Figure 1.2: A generalization of puncturing: a serial concatenation of a puncturing code of rate \( \geq 1 \) and a mother code

In a Turbo-coding setting (with a parallel concatenation), such structures lead to hybrid Turbo codes, where two (or more) inner serial concatenations are again concatenated in a parallel fashion, which leads to scheduling issues and questions regarding the transfer of EXIT charts, which will be studied more for the pruning case in Section 1.1.2.

\(^1\)Fractions of polynomials for recursive encoders.
Rate compatible punctured convolutional codes [4] ensure that the rate change does not lead to adverse effects on the free distance of the convolutional code. Puncturing matrices are built on each other, i.e., adding more punctured bits to already existing ones, avoiding permutations in the pruning pattern for different code rates.

To summarize, puncturing allows to increase the code rate, thereby weakening the codes (lower free distance).

Pruning is the opposite procedure, where in its simple form, input bits are omitted, thereby increasing the code rate of a mother code. In a general setting, we obtain a serial concatenation with a swapped order compared to Fig. 1.2.

In a Turbo-code setting, this again leads to a hybrid parallel/serial concatenation studied in Section 1.1.2.

Let us consider an example of two codes with generator matrices

\[
G^{(0)}(D) = \begin{bmatrix}
1 & 1 + D & 1 + D \\
1 + D & D & 1 + D
\end{bmatrix} \quad \text{and} \quad G^{(1)}(D) = \begin{bmatrix}
1 + D + D^2 & 1 + D + D^2 & 1 + D^2
\end{bmatrix},
\]

where the second can be considered as a sub-code of the first. It is obvious that a pruning matrix

\[
P(D) = \begin{bmatrix}
1 & D
\end{bmatrix}
\]

multiplied with \(G^{(0)}(D)\) from the left leads to \(G^{(1)}(D)\).

The trellis segments of both codes \(G^{(0)}\) and \(G^{(1)}\) are shown in Fig. 1.4. Pruning means discarding trajectories from the trellis of \(G^{(0)}\) to realize the subcode \(G^{(1)}\).

Regarding the rates, we observe that the number of input bits is reduced
1.1. UNEQUAL ERROR PROTECTION TURBO CODES

From $k^{(0)}$ to $k^{(1)}$, i.e., the rate of the pruned code is given by

$$R^{(1)} = \frac{k^{(1)}}{n} = \frac{k^{(1)}}{k^{(0)}} \cdot \frac{k^{(0)}}{n} = R_P \cdot R^{(0)}; \quad (1.4)$$

with $R_P$ and $R^{(0)}$ being the rates of the pruning and mother codes, respectively.

In a more strict sense, just like with puncturing understood as omitting output data, pruning can be seen as omitting input data, i.e., replacing input data by known values, e.g., by zeros according to some pruning scheme as shown in Fig. 1.5. In a soft decoder, pruned input bits are exactly known, i.e., represented by log-likelihood ratios of $\pm \infty$ for a $\pm 1$ representation.

Table 1.1 summarizes the rates achievable by omitting $p$ bits at the output (puncturing) or input (pruning).

Figure 1.4: Trellis segments of an unpruned (left) and a pruned (right) code

Figure 1.5: Pruning by inserting known values, e.g., zeros instead of data according to some pruning pattern
Table 1.1: Puncturing and pruning rates

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>puncturing</td>
<td>( R = \frac{k}{n-p} )</td>
</tr>
<tr>
<td>pruning</td>
<td>( R = \frac{k-p}{n} )</td>
</tr>
<tr>
<td>non-systematic</td>
<td></td>
</tr>
<tr>
<td>bits</td>
<td></td>
</tr>
<tr>
<td>systematic</td>
<td></td>
</tr>
<tr>
<td>bits</td>
<td></td>
</tr>
</tbody>
</table>

The application of puncturing and/or pruning to Turbo codes follows in the natural way. Puncturing has been studied, e.g., in [18–20], pruning in [21, 22]. Reference [23] discusses bounds for UEP Turbo coding.

1.1.2 Hybrid Turbo codes and their convergence

Turbo codes [24] were originally defined as parallel concatenated codes. Later, Benedetto et al. [25] introduced a serial concatenation of interleaved convolutional codes which in general exhibit lower error floors than parallel concatenated codes, but usually converge further away from channel capacity. Hybrid concatenated codes offer a combination of parallel and serial concatenations, with the opportunity to exploit both the advantages of parallel and serially concatenated codes. Among the different hybrid concatenated structures proposed in literature [26, 27], we focus herein on the hybrid scheme depicted in Fig. 1.6 [28]. This kind of concatenation consists of a parallel concatenation of two serially concatenated interleaved codes and arise in the context of Turbo coding for UEP applications [21, 22].

Figure 1.6: Encoder structure of a hybrid turbo code.
1.1. UNEQUAL ERROR PROTECTION TURBO CODES

The role of hybrid concatenated codes in UEP applications are well described in [29] where the authors showed that a pruning procedure can be employed to adapt the rate and distance for different protection levels in UEP Turbo codes. Moreover, the authors show that pruning of parallel concatenated convolutional codes can be accomplished through the concatenation of a mother code and a pruning code. For example, in Fig. 1.6, the codes $G_{11}$ and $G_{21}$ can be referred to as the pruning codes and $G_{12}$ and $G_{22}$ as the mother codes of the pruning scheme.

The decoding of hybrid Turbo codes is divided into a local decoding corresponding to each parallel branch, and a global decoding step where the parallel branches exchange extrinsic information between them. As a tool to investigate the iterative decoding behavior of the hybrid concatenation, we define two different EXIT charts [30]: local and global. The local charts examine the iterative decoding behavior of each parallel branch separately. The global EXIT chart deals with the exchange of information between each parallel branch during the decoding procedure. Both local and global charts provide good insight into the convergence behavior of hybrid Turbo codes. In what follows, we define and show how to construct the local EXIT charts and how those relate to the EXIT charts of parallel concatenated convolutional codes previously studied.

In this section, we will assume that the codes shown in Fig. 1.6 are recursive systematic convolutional codes with rates $R_{11} = R_{21} = 1/2$ and $R_{12} = R_{22} = 2/3$. The generator polynomials of the example codes are given by

$$G_{11} = G_{21} = \left( 1 \frac{D^2}{1+D+D^2} \right) \quad (1.5)$$

and

$$G_{12} = G_{22} = \left( \begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right) \frac{1+D+D^2}{1+D+D^2} \quad (1.6)$$

The systematic coded bit stream is formed as follows

$$c = (c_1 \ c_2) = (c_{1,1}(1) \ c_{1,2}(1) \ c_{1,3}(1) \ c_{2,2}(1) \ c_{2,3}(1))$$

$$\quad c_{1,1}(2) \ c_{1,2}(2) \ c_{1,3}(2) \ c_{2,2}(2) \ c_{2,3}(2) \ldots ,$$

where $c_{1,1}(1) = u(1)$, $c_{1,1}(2) = u(2)$ and so on. In order to keep the overall system systematic, only the parity bits of $x_1$ and $x_2$ are interleaved in the serial concatenations. Note that $c_{2,1}(.)$ is not transmitted, since we do not want to transmit the systematic information twice. The overall rate of our example code is $R = 1/5$. 
Local EXIT charts

The convergence of the parallel branches under iterative decoding can be investigated by means of what we call local EXIT charts. Local EXIT charts are a generalization of the graphic depiction of the transfer characteristics of a serial concatenated coding scheme [31]. The difference between local EXIT charts and the charts for interleaved serial concatenated convolutional codes is that the former is composed not of two curves (outer and inner code information transfer characteristics), but of the transfer curve of the inner code and a set of curves representing the evolution of the information transfer characteristic of the outer decoders as the iterative decoding is performed.

The necessity of representing the transfer characteristic of the outer codes of each branch arises from the fact that in the end of each local (serial) decoding, one parallel branch sends information concerning the uncoded bits to its adjacent branch. Figure 1.7 illustrates this situation. The vertical axis denotes the extrinsic (a priori) information of the inner (outer) decoder $I_{e,i}$ ($I_{a,o}$) and the horizontal axis denotes the a priori (extrinsic) information of the inner (outer) decoder $I_{a,i}$ ($I_{e,o}$).

The solid and the dashed thin lines represent the transfer characteristic of the upper and lower outer decoder, respectively. The bold line represents the transfer characteristic of the inner decoder which remains unchanged during the decoding procedure since the a priori information received by the inner decoder is the channel (intrinsic) information which remains constant during the whole decoding.

Note that the transfer characteristic of the outer decoders starts at the abscissa zero (first local decoding operation) and is shifted to the right at the beginning of each further local decoding. As mentioned before, this shift is a consequence of the fact that at each local iteration, new information regarding the systematic bits ($I_a(\hat{u})$) is received from the adjacent parallel branch. This set of information transfer curves of the outer decoder for different $I_a(\hat{u})$ (together with the transfer curve of the inner decoder) is the local EXIT chart.

Let $l_1$ and $l_2$ be the number of local iterations of the upper and lower branches, respectively. Furthermore, let a global iteration be defined as the process composed of the set of $l_1 + l_2$ local iterations and the exchange of information between the parallel branches. In Fig. 1.7, each global iteration is represented by a pair of curves for the outer decoders (a dashed and solid line pair). The convergence of the decoding can be inferred from the local EXIT
1.1. Unequal Error Protection Turbo Codes

Figure 1.7: Local EXIT chart of the example hybrid Turbo code for $E_b/N_0 = -1.22 \text{ dB}$. For this SNR, it can be noticed from this chart that the system will converge when the number of global iterations is greater than 3.

chart by the existence of an “open tunnel” between the transfer characteristic of the inner and outer decoders. For example, for the local EXIT chart of Fig. 1.7, there is an “open tunnel” for more than two global iterations which means that the iterative decoding of the hybrid Turbo code will converge for the depicted SNR when more than 3 global iterations are considered.

Relation between local and global EXIT charts

The convergence analysis of hybrid Turbo codes subject to iterative decoding can be investigated by means of both local and global EXIT charts. Since both types of charts lead to the same conclusion regarding the convergence of the decoding, it is expected that there is a mathematical relation between them. The first step towards the derivation of such a relation is to identify the different quantities depicted in each chart.

Local EXIT charts depict the relation between the a priori and extrinsic information concerning the output of the outer decoder, i.e., $I_{a,o}(\hat{x})$ and $I_{e,o}(\hat{x})$, where we use $\hat{x}$ to denote the estimate of the decoder regarding the codeword $x$. Global EXIT charts, instead, depict the relation between the a
CHAPTER 1. RATE-COMPATIBLE LDPC AND TURBO CODES FOR LINK ADAPTIVITY

priori and extrinsic mutual information values regarding the systematic bits, i.e., $I_{a,j}(\hat{u})$ and $I_{e,j}(\hat{u})$ where $j = 1$ (upper branch) or 2 (lower branch).

Note now that the points of zero ordinate in the local EXIT charts ($I_{a,o}(\hat{x}) = 0$) indicate the absence of a priori information regarding the output bits of the outer encoder. At these points, all the information about $\hat{x}$ that the outer decoder has is obtained from the information regarding the systematic bits $\hat{u}$, which is provided by the other parallel branch. Given that the outer code is systematic, we can write $x = [u, p]$, where $p$ denotes the parity bits resulting from the encoding of $u$ by the outer encoder. Moreover, noticing that the information regarding the parity bits $p$ is zero in the points of zero ordinate, we can write

$$I_{e,o}(\hat{x}) = R_o \cdot I_{e,j}(\hat{u}) + (1 - R_o) \cdot I_{e,j}(\hat{p}) = R_o \cdot I_{e,j}(\hat{u}) ,$$

(1.7)

where $R_o$ is the rate of the outer code, and $j = 1$ or 2 depending whether the upper or lower decoder was activated in the corresponding local decoding.

A closer analysis of Eq. (1.7) reveals that it provides a link between local and global EXIT charts, since it relates quantities that are depicted in both types of charts. On the one hand, we can compute the global decoding trajectory from the points where $I_{a,o}(\hat{x}) = 0$, since all the information that the outer decoder has about $\hat{x}$ at these points comes from the information regarding the systematic bits $\hat{u}$. Thus, applying Eq. (1.7), we can compute the corresponding $I_{e,j}(\hat{u})$. On the other hand, note that the points in the global EXIT chart where the decoding trajectory and the transfer curve of the active branch meet correspond to the instant in the global decoding where the parallel branches exchange information regarding the systematic bits $\hat{u}$, this means that using those points, we can compute the abscissas of the local EXIT chart where $I_{a,o}(\hat{x}) = 0$.

Fig. 1.8 depicts how to relate the local and global EXIT charts by means of Eq. (1.7). Those charts were constructed using our example codes and $E_b/N_0 = 1$ dB. Note that since the outer code rate is $R_o = 0.5$, we can write $I_{e,o}(\hat{x}) = 0.5 \cdot I_{e,j}(\hat{u})$, where $j = 1$ for the upper branch (dashed arrows) and $j = 2$ for the lower one (solid arrows).

1.1.3 Interleaver structures

For Turbo codes, typically, the so-called $S$-random interleaver [32, 33] is applied. Other options are algebraic ones which are more easily specified, but
1.1. UNEQUAL ERROR PROTECTION TURBO CODES

Figure 1.8: Relation between the points $I_{a,o}(\hat{x}) = 0$ in the local EXIT chart and the global decoding trajectory for the example hybrid Turbo code. The charts were constructed for $E_b/N_0 = 1$ dB (related to the global system) with the lower decoder being activated first.
have performance disadvantages. Such constructions [34, 35] are related to the linear congruential method of pseudo-random number generation. Interleavers have to be somewhat separated for the individual priority classes such as in [18]. Separating it completely, especially for low-rate components, e.g., obtained by pruning, means a short individual interleaver for the corresponding data. One may strictly separate the data of different priority classes or allow for some leakage between the classes, whereas the latter will then, of course, somewhat worsen the error performance of the highest priority class. Some leakage will, however, interconnect data thereby realizing a bigger interleaver, improving the average performance. [36] introduced the relaxation of the class boundaries permuting a limited portion of data between the priority classes. Strictly separated equalizers should, of course, be optimized for the given length. [37] discusses this for the example of blocked convolutional interleavers. However, convolutional interleavers do not necessarily need to be blocked, if one would go for a pipelined decoder structure. One can actually also randomize the convolutional interleaver according to some \( m \)-sequence or alike.

\( S \)-random interleaver is a random interleaver with an extra constraint. The \( S \)-random interleaver or Fixed Block \( S \)-Random interleaving algorithm is as follows:

1. Given \( N \) integers, randomly select one out of the integer pool without replacement.
2. Check if integer is outside the range \( \pm S \) of \( S \) past values. If outside the range, keep the value, else, reject it and place it back into the integer pool.
3. Repeat the steps until no integers in range \( 1 \) to \( N \) are left unused.

The algorithm ensures a certain minimum spread realized by the interleaver. Integers \( i \) and \( j \) satisfying \( |i - j| < S \) will be reshuffled such that \( |\pi(i) - \pi(j)| > S \) after interleaving [38] (for short interleavers and multiple Turbo codes, see the works [39] and [40], respectively). The convergence limit is \( S < \sqrt{N/2} \). For larger interleavers, \( S \) may be significantly lower [41].

Figure 1.9 (a) outlines the strictly separating interleaver, which means a block-diagonal permutation matrix (ones marked as dots) and (b) shows the one allowing for some leakage.
1.1. UNEQUAL ERROR PROTECTION TURBO CODES

Figure 1.9: Overall interleaver as a block-diagonal matrix (a) without and (b) with leakage

Denoting the interleaving by $\pi$ and the leakage to other priority classes by $\gamma$, one may realize operation sequences $\pi \rightarrow \gamma$ or $\gamma \rightarrow \pi$, i.e., first interleaving and then leakage or vice versa. With leakage, one has, of course, to ensure that bits moved outside a certain priority class are replace by others. Figure 1.10 shows the trivial case, where bits from an intermediate priority class are permuted with bits from the neighboring ones.

Figure 1.10: Example for leaking interleaver map

Figures 1.11 and 1.12 provide individual class performances and average performances, respectively (3 equal-size interleaver areas of length 1000). The first shows a narrowing of the performance spread between classes, but also a flooring towards the next worse class due to the influence of bits moved into the worse protected neighboring class. The improvement of the average performance with increased leakage is in line with the improved performance of the weakest class due to bits moved into the better protected class.

As described in Section 1.1.1, puncturing and pruning schemes mean setting log-likelihood ratios to 0 or $\pm \infty$, leading to irregularities in the intrinsic information for the decoding. [42] discusses these issues and chooses symmetric mod-$k$ interleavers to level out the effects.
Figure 1.11: Turbo code class performances with and without leaking interleavers

Figure 1.12: Turbo code average performance with and without leaking interleavers
1.2 Unequal Error Protection LDPC Codes based on Puncturing and Pruning

Low-density parity-check (LDPC) codes are a class of powerful error correcting codes that were first introduced in the seminal work of Gallager [43] in 1963. They were rediscovered in the 90’s, where design and implementation techniques for these codes were researched intensively and several powerful codes were discovered. LDPC codes form part of various industrial standards and are widely used in commercial chip sets today.

A binary LDPC code is defined by a binary parity-check matrix $H$ of dimension $(N - K) \times N$, where $N$ represents the block length of the code and $K$ represents the number of information bits. The set of valid codewords is given by the set $C = \{x : Hx = 0\}$. In LDPC codes, the parity-check matrix $H$ is sparse, i.e., the number of ones in each row and column is much smaller than the row or column dimension. The sparsity of $H$ helps to significantly simplify the decoding of these codes. The parity-check matrix can also be viewed by considering its Tanner graph representation. Specifically, $H$ is in one-to-one correspondence with a bipartite graph $T = (V \cup C, E)$. The nodes in $V$ are referred to as the variable nodes and correspond to the codeword symbols (columns of $H$); thus, $|V| = N$. The nodes in $C$ are referred to as the check nodes and correspond to the parity check equations (rows of $H$); thus, $|C| = N - K$. Let $v \in V$ and $c \in C$. There exists an edge between $v$ and $c$ if the codeword symbol corresponding to $v$ participates in the parity check equation corresponding to $c$. It is evident therefore that $H$ and $T$ are in one-to-one correspondence (see Fig. 1.13 for an example).

LDPC codes are decoded using the message passing algorithms (MPA) [44]. Several variants of the basic MPA are known including the sum-product, min-sum etc. (see [45]). The rediscovery of LDPC codes in the 90’s [46] and techniques for the design of LDPC codes were the focus of several influential papers [47–52]. It turns out that in the asymptotic setting (when one considers $N \to \infty$), the error correction properties of LDPC codes only depend on the degree sequence of $T$. In particular, when $N \to \infty$ for a given channel (such as the BSC or AWGN), one can determine a channel parameter threshold, such that with high probability, the code is guaranteed to correct all errors as long as the channel parameter is below the threshold. Conversely, when the channel parameter is above the threshold, the decoding will lead to erroneous results with high probability. The original work
of Gallager considered regular LDPC codes, where the number of ones in each row and column was a constant and determined thresholds for various channels for such codes. For instance a $(3, 6)$ regular LDPC code is such that each variable node has degree 3 and each check node has degree 6. The work of [48, 52] considered the potential improvement in the threshold by considering irregular degree sequences where there are multiple possibilities for the left- and right-node degrees. Specifically, it was shown that a careful choice of these degree sequences could significantly boost the threshold of LDPC codes. Several techniques for constructing LDPC codes that have good BER performance in the finite-length regime have also been investigated [53–59].

There are also a large number of classes of sparse graph codes that have been examined in the literature which can be considered as subclasses of LDPC codes. These include various classes of repeat-accumulate (RA) codes [60–62], protograph codes [63–66], quasi-cyclic LDPC codes [67] etc. From an implementation point of view, there are several issues that need to be considered when using LDPC codes. For instance, when considering ASIC implementations, the storage space for the parity-check matrix needs to be low. A randomly chosen LDPC code from a good degree distribution may have a good threshold but a high storage cost. Practical implementation constraints thus make structured LDPC codes attractive. For instance, the basic representation of a protograph based LDPC code can be specified by a small protograph along with appropriately chosen circulant matrices [65,68]. In this survey, we will consider rate compatibility issues in several subclasses of LDPC codes and structured LDPC codes as well.
1.2. UEP LDPC CODES BASED ON PUNCTURING AND PRUNING

The basic concept of rate-compatibility as introduced above for Turbo codes applies to LDPC codes as well. Namely, upon encoding, certain parity bits are not transmitted (these are the “punctured” bits); the receiver attempts to decode by treating the untransmitted bits as erasures. Significant research work has addressed various aspects of rate-compatible LDPC codes. We now proceed to discuss the issues pertaining to rate-compatible LDPC codes in more detail.

1.2.1 Density evolution for general RC-LDPC codes

As noted above, in the limit of large block length $N \to \infty$, the properties of LDPC codes essentially depend only on the degree sequences of the variable nodes and the check nodes. Specifically, we represent the edge degree sequence of the variable nodes by a polynomial $\lambda(x) = \sum_{i=1}^{d_l} \lambda_i x^{i-1}$ and the edge degree sequence of the check nodes by a polynomial $\rho(x) = \sum_{i=1}^{d_r} \rho_i x^{i-1}$. Here, $d_l(d_r)$ is the maximum left (right) node degree and $\lambda_i(\rho_i)$ represents the fraction of edges connected to variable (check) nodes of degree $i$. The work of [49, 52] proposed the technique of density evolution that allows one to determine the code threshold for various channel models. As the description of density evolution is rather involved, we will not go into this aspect in detail; however, we discuss the major results that have appeared in the literature.

For a given “mother code” with degree distribution $(\lambda(x), \rho(x))$, the work of [69] presents techniques for the design of efficient puncturing distributions $\pi(x)$, whereby a fraction $\pi_i$ of the variable nodes of degree $i$ are punctured. Specifically, [69] leverages the Gaussian-approximation-based density evolution [50] to determine the optimal $\pi(x)$ and specific examples can be found in Tables I-III of [69].

Another useful technique for determining asymptotic thresholds is the so-called EXIT chart technique [30] which was first proposed for understanding the convergence behavior of iteratively decoded parallel concatenated codes, and were later generalized to the analysis of LDPC codes [70–73]. The components of an EXIT chart are the EXIT functions of the constituent code components of the iterative decoder, which relates the a priori mutual information available to a code component, denoted by $I_A$ and the extrinsic mutual information generated after decoding, denoted by $I_E$. The advantage of EXIT charts is that the code design problem can be reduced to a curve fitting problem between the code components (usually two in num-
CHAPTER 1. RATE-COMPATIBLE LDPC AND TURBO CODES FOR LINK ADAPTIVITY

ber). The EXIT chart technique is especially useful in situations where there are structured parts in the codes, e.g., IRA (irregular repeat-accumulate) and protograph codes. One can generate EXIT curves for code components where some of the nodes are punctured (see [70,74]). This helps to determine the code thresholds in an efficient manner.

1.2.2 Design of good puncturing patterns for a given mother code

We now focus on the design of finite length RC-LDPC codes (and corresponding puncturing patterns) that have good BER performance. For a given mother code, an immediate question is the determination of the parity bits that should be punctured for obtaining rate-compatibility. Note that one typically needs a range of rates, e.g., starting from a mother code rate of 0.5, one may need rates in steps of 0.1 with a maximum code rate of 0.9. It is also important the set of parity bits to be nested, as practical systems typically operate in an incremental redundancy mode where the transmitter attempts to operate at a high code rate first and then sends additional parity bits based on feedback from the receiver. Several techniques have been developed in the literature for the design of appropriate puncturing patterns.

For a given puncturing pattern, the work of Ha et al. [75] defines the concept of $\beta$-step recoverability and uses it to find optimized puncturing patterns. For a given puncturing pattern, let $V_0 \subset V$ represent the set of unpunctured nodes. Thus, the set of punctured nodes is given by $V \setminus V_0$. A punctured node $p \in V \setminus V_0$ is called 1-step recoverable (1-SR) if there exists a check node $c$ such that $p$ is the only punctured node connected to $c$ (see Fig. 1.14).

It can be observed that if the channel does not induce any errors, then 1-SR nodes will be recovered in one iteration of iterative decoding. Intuitively, in the high-SNR regime, we expect a similar effect, i.e., 1-SR nodes will be recovered somewhat easily under iterative decoding. There is a natural extension of the concept of 1-SR nodes to $\beta$-SR nodes where $\beta > 1$ (see [75] for details). The main idea of [75] is a greedy approach that attempts to maximize the number of 1-SR nodes that can be found for a given mother code. The algorithm proceeds in stages, where it transitions to finding 2-SR nodes, once the number of 1-SR nodes is saturated, and then 3-SR nodes and so on until the maximum code rate is reached. Reference [75] demonstrates
1.2. UEP LDPC Codes Based on Puncturing and Pruning

Figure 1.14: Filled circles represent unpunctured variable nodes and unfilled circles represent punctured variable nodes. Note that \( v \) has one check node neighbor that is connected to all unpunctured nodes except \( v \). Thus, \( v \) is a 1-SR node.

Punctured codes that have a low performance gap (\( \approx 0.2 \) dB) of dedicated LDPC codes using this approach. This basic idea of [75] was modified and improved upon in different ways in the work of [76–79]. The work of [80, 81] (see also [82]) proposes ranking algorithms for shortlisting good puncturing patterns for a mother code using Gaussian-approximation-based density evolution techniques. This also results in patterns with good performance across a range of rates.

1.2.3 Pruning for creating irregular UEP check-node profiles

Usually, UEP LDPC codes are constructed focusing on the variable-node distribution, increasing the degree at high-priority variable nodes. However, considering the check-node side, lower degrees of check nodes improve the performance of variable nodes connected to them. Hence, one can instead also reduce the average check-node degree of such check nodes connected to high-priority variable nodes.

In [83, 84], an iterative pruning approach is presented, omitting variable nodes of a systematic part of the codeword. Thereby the average check-node degree of higher-priority variable nodes is lowered, since pruned variable nodes will in return lead to the omission of corresponding edges (see Fig. 1.15), noting that pruning means knowing the corresponding information at such variable nodes, in contrast to puncturing, where the value is unknown. This means, pruning leads to \( \pm \infty \) for the log-likelihood ratios depending on the actual bit value, which one might choose to be equally
distributed, but known. In the mentioned papers, an iterative pruning algorithm is described starting from given variable and check node distributions, modifying the rate and check-node distribution such that a certain priority profile is obtained. To obtain a systematic representation, a triangular parity-check matrix is used, stepwise pruning variable nodes in the non-triangular, systematic part, thereby erasing corresponding columns from the matrix. Starting from a mother code with rate $K_0/N_0$, a daughter code of rate $K_1/N_1$ is obtained by pruning $K_0 - K_1$ columns, thereby also reducing the length to $N_1 = N_0 - (K_0 - K_1)$. Apart from typical conditions, such as convergence, stability, rate, additional constraints have to be fulfilled. There is a lower limit of the check-node degree, and one has to avoid unvoluntary pruning that could make a column of the parity-check matrix independent of all others. The resulting degree distribution is not necessarily concentrated any more.

![Figure 1.15: Pruning of an LDPC code](image)

In general, as already described in Section 1.1.1 for convolutional codes, pruning can also be realized by a precoding, which is not discussed for LDPC in here. Furthermore, puncturing can also be combined with pruning for higher flexibility.

### 1.2.4 Structured RC-LDPC codes

In practice, most LDPC codes are not chosen from an arbitrary degree distribution profile. There are several reasons for this. As mentioned above, there are classes of LDPC codes that have certain desirable characteristics, e.g., repeat-accumulate codes and their variants can be encoded in linear-time. Moreover, in the specific context of rate-compatible puncturing, for performance reasons it is often useful to design the part of the parity-check matrix that corresponds to parity bits in a structured manner.
1.2. UEP LDPC CODES BASED ON PUNCTURING AND PRUNING

There are several proposed constructions for RC-LDPC codes that have structured parts. The work of [85] proposes a class of codes called $E^2RC$ codes that have a specific lower-triangular structure for the parity part. In particular, there is no cycle that consists of exclusively parity nodes in the corresponding Tanner graph. Furthermore, approximately half of the parity nodes are 1-SR, one-fourth are 2-SR, and so on. For example, consider the following $8 \times 8$ binary matrix that is constructed using the $E^2RC$ technique.

$$H_2 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1
\end{bmatrix}$$

It is evident that if the parity nodes of a LDPC code correspond to the columns of $H_2$, then there does not exist a cycle among them. Moreover, the first four columns correspond to 1-SR nodes, the next two correspond to 2-SR nodes and the seventh and eighth column correspond to a 3-SR and 4-SR node. The $E^2RC$ structure is shown to have superior performance under rate-compatible puncturing across the range of rates. The work of [74] extends this class of codes by providing systematic techniques based on EXIT charts to optimize the degree distribution of the systematic nodes under the constraint that the parity part has the $E^2RC$ structure. The resulting codes are at most 0.35 dB away from the capacity limit across the range of rates. Reference [86] provides a method to reduce the error floor of $E^2RC$ codes. The work of [87] develops puncturing algorithms and determines thresholds for IRA codes. Starting from a given mother LDPC code, the work of [88] proposes deterministic structures for extending and puncturing it.

Implementation constraints often dictate that the amount of storage used to represent the parity check matrix be small. Hence, a very popular technique for constructing LDPC codes in practice is to start with a small graph called the protograph. Each edge in the graph is replaced with a permutation (which is usually a circulant permutation) of an appropriate size. For instance, if the protograph has $N_{pr}$ variable nodes and $M_{pr}$ check nodes, and the permutations are of order $p$, the resultant code is of blocklength $pN_{pr}$ and has $pM_{pr}$ check nodes. The idea of protograph LDPC codes was first
introduced in the work of [63]. The key advantage of this methodology is that one can focus on the design of the protograph. Specifically, it turns out that assuming that the edges are replaced with protographs of sufficiently large order, one can perform density evolution (by leveraging the multi-edge LDPC framework [89]) and determine the threshold of the code based on the protograph alone. Furthermore, one can use techniques such as PEG [53] and ACE [54] for the choice of the permutations. A similar strategy applies to the case of rate compatible protograph LDPC codes. Given a mother protograph, one can choose an appropriate puncturing pattern for the protograph itself and evaluate the different patterns using density evolution. Such a strategy has resulted in the development of a large number of code families with very good performance across the range of rates [66].

There is a large class of RC-LDPC codes that are obtained by using protographs. In principle, one starts from a “mother” protograph and then identifies certain parity variable nodes in it that are suitable for puncturing. One can easily determine the threshold of the various protographs via density evolution by using the multi-edge framework introduced in [89]. Some works [74, 90] have also used the technique of check node splitting that instead starts from a high rate protograph and iteratively splits the check nodes to produce a rate-compatible family of lower rate protographs. Among the first works to address the design of protograph families was [91], which presents several families of protographs (AR34A, AR4A, ARJA) that are essentially hand-designed and provide a good performance across the range of rates. This was followed up with the work of [66] that presented a family of codes called the AR4JA family with rates from 1/2 to 7/8 and thresholds gaps to capacity at most 0.441 dB. The work of [74] presented a class of protograph codes that was inspired by the $E^2RC$ codes. Specifically, the search for good protographs was computer guided and resulted in codes where the gap to capacity was at most 0.3 dB. More recently, the work of [92] demonstrates families of protographs with even better performance.

1.3 Unequal Error Protection LDPC Codes based on degree distribution optimization

A typical approach to design LDPC codes for UEP applications relies on the optimization of the variable node degree distribution of the code. The
reasoning of this approach is based on the observation that the connection degree of a variable node affects the error rate of the symbol it represents, i.e., variable nodes with higher connection degree have lower error rates. Starting from this observation, the use of irregular LDPC codes for UEP applications was investigated in [93,94]. Herein, we focus on the UEP LDPC codes introduced by Pouliiat et al. in [93] and discuss how to enhance its UEP capabilities by means of a multi-edge optimization [10].

The basic idea to optimize the degree distributions of irregular LDPC codes in order to provide them with UEP capability is defining local variable degree distributions, i.e., each protection class defined among the symbol nodes is described by a polynomial $\lambda^{(j)}(x) = \sum_{i=2}^{d_{\text{max}}^{(j)}} \lambda^{(j)}_{i} x^{i-1}$, where $\lambda^{(j)}_{i}$ represents the fraction of edges connected to degree $i$ variable nodes within the protection class $C_{j}$. In [93], the authors observed that the error rate of a given protection class depends on the average connection degree and on the minimum degree of its variable nodes, $d_{v_{\text{min}}}^{(j)}$. Accordingly, they proposed an optimization algorithm where the cost function is the maximization of the average variable node degree subject to a minimum variable node degree $d_{v_{\text{min}}}^{(j)}$. As in [93], we interpret the unequal error protection properties of an LDPC code as different local convergence speeds, i.e., the more protected a class is, the faster is its convergence to its right value.

An important characteristic of the UEP LDPC codes constructed in [93] is that the performance difference between the protection classes defined within a codeword is strongly dependent on the level of connection between them. That is, the more interconnected two protection classes are, the smaller will be the difference between their performance. This characteristic motivated us to investigate the application of a multi-edge-type analysis of UEP LDPC codes, since it allows to distinguish the messages originating from different protection classes. This ability to distinguish the messages according to their originating protection class provide us with the means to adjust the connectivity among the different classes, thus controlling the difference between their performance. In the following, we explain shortly how a multi-edge framework can be applied to enhance the UEP performance of an LDPC code. For a more detailed analysis of the optimization algorithms for the degree distributions and connectivity profile between the protection classes, the reader is referred to [93] and [10].
1.3.1 Multi-edge-type UEP LDPC codes

First introduced in [9], multi-edge-type LDPC codes are a generalization of irregular and regular LDPC codes where several edge classes can be defined and every node is characterized by the number of connections to edges of each class.

Unequal error protection LDPC codes can be included in a multi-edge framework in a straightforward way. This can be done by distinguishing between the edges connected to the different protection classes defined within a codeword. According to this strategy, the edges connected to variable nodes within a protection class are all of the same type. Consider for example Fig. 1.16.

![Multi-edge factor graph with two different edge types](image)

Figure 1.16: Multi-edge factor graph with two different edge types

In this figure, the first 4 variable nodes are assumed to belong to the same protection class, thus the edges connected to them are defined as type-1 edges (red lines). The last 3 variable nodes form another protection class, thus the edges connected to them are defined as type-2 edges (blue lines). Note that as opposed to the variable nodes, the check nodes admit connections with edges of different types simultaneously. Based on this multi-edge framework, [10] derives an optimization algorithm for the connection profile of the UEP LDPC codes obtained in [93].

The multi-edge optimization of the connectivity among the protection classes is a linear optimization problem with cost function defined as the minimization of the average check node degree of the less protected classes. The algorithm proceeds sequentially optimizing the check node degree distributions of each class proceeding from the least protected class to the most protected one.

---

2A similar multi-edge framework can also be used to the analysis of UEP rateless codes [11].
1.3. UEP LDPC CODES BASED ON DEGREE DISTRIBUTION OPTIMIZATION

This scheduling minimizes the amount of extrinsic information originating from the less protected classes that is received by the more protected ones, i.e., the check nodes should have a minimum number of connections to the less protected classes in order to avoid that unreliable messages are forwarded to better protected ones.

Figure 1.17 compares the performance of an UEP LDPC code with 2 protection classes before and after the optimization of its connectivity profile (referred to as UEP and MET UEP, respectively). Both codes have rate $R = 1/2$ and length $N = 4096$ bits. All simulations were performed considering binary modulated symbols transmitted over an AWGN channel and a total of 7 decoding iterations. As a benchmark, Fig. 1.17 still depicts the performance of a code of same size and rate optimized for the AWGN without any UEP constraints (referred to as AWGN). Note that both protection classes of the MET UEP code have a better performance than the UEP and AWGN codes for the considered signal-to-noise ratio range and number of decoding iterations.
Bibliography

coding extension of the H.264/AVC standard. Circuits and Systems for

transmission of data partitioned H.264 video with hierarchical QAM.


codes) and their applications. Communications, IEEE Transactions on,

Viterbi and sequential decoding. Communications, IEEE Transactions on,


[37] Sina Vafi, Tadeusz A. Wysocki, and Ian Burnett. Convolutional interleaver for unequal error protection of turbo codes. In *Proceedings of


