A Study on Design of Error-correcting Schemes Using Graph Representations

グラフ表現を利用した誤り訂正方式の構成に関する研究

March 2009

Naoto Kobayashi

Contents

Chapter 1

Introduction

1.1 Digital Data Transmission and Storage

In recent years, the demand for efficient and reliable digital data transmission and storage system has been increasing. When digital data is transmitted through a communication system or written into a storage system, the systems generate noise. This noise may result in the occurrence of an error in the digital data. It is necessary to correct the error if a noise-corrupted digital data is received by the destination or read from a storage system. We must consider a process in which the receiver is capable of correcting the error in the digital data.

Digital data transmission and storage are simplified communication between a transmitter and a receiver through a noisy *channel*, as shown in Fig. 1.1

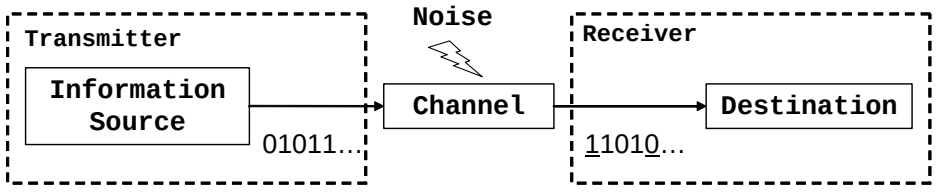

Fig. 1.1 Simplified model of digital data transmission and storage system.

1.2 Error-correcting Codes

Error-correcting codes enables the correction of the error while increasing the size of digital data to be transmitted or stored. In recent years, error-correcting codes have been widely used in data transmission and in storage systems. An error-correcting procedure is shown in Fig. 1.2.

An information source emits a digital signal called an information symbol. The information symbol is represented by a member of the finite set $\mathscr{U}$. Let us consider a sequence of information symbols with a finite length, called an *information sequence*, which is denoted by $\mathbf{u}$.

Let $\mathscr{X}$ be a finite set with q elements. Consider a mapping from the information sequence

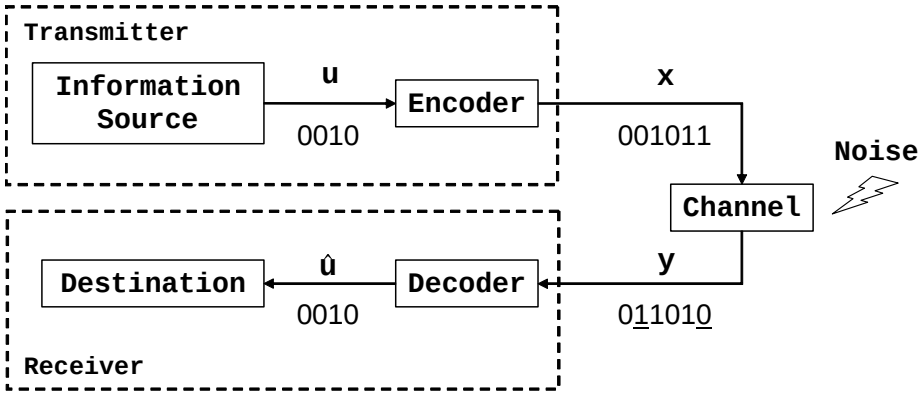

Fig. 1.2 Error correction, where $\mathscr{U} = \mathscr{X} = \mathscr{Y} = \{0, 1\}$.

to a sequence of elements in $\mathscr{X}$. Such a mapping is referred to as an *encoder* and a sequence within the given mapping range is called a *codeword*, which is denoted by $\mathbf{x}$. The symbol of the codeword is called a codeword symbol. The range is called a *code*.

The transmitter encodes the information sequence $\mathbf{u}$ to the codeword $\mathbf{x}$ by using the encoder and transmits $\mathbf{x}$ through the channel. Once $\mathbf{x}$ is transmitted through the channel, the receiver receives a noise-corrupted codeword $\mathbf{y}$, which is called a received sequence. The symbol of the received sequence is called a received symbol. The received symbol is represented by a member of the finite set $\mathscr{Y}$.

The receiver estimates the transmitted information sequence by using the given received sequence. Therefore, consider a mapping from the received sequence to the estimator of the information sequence; this mapping is referred to as a *decoder*. The estimator of the information sequence is denoted by $\hat{\mathbf{u}}$.

Assume that the information symbols occur according to a probability distribution, and assume that the channel is governed by a probability distribution of an output sequence given an input sequence. Then, the above error-correcting procedure is represented by a probabilistic model of which each symbol is represented by a random variable.

The redundancy due to the encoding process is represented by a measure, *code rate*. A block error is said to occur when $\mathbf{u} \neq \hat{\mathbf{u}}$, and the probability of the occurrence of the block error is called block error rate. Similarly, a symbol error is said to occur when an information symbol does not equal to the corresponding estimate of the information symbol. The probability of the occurrence of the symbol error is called symbol error rate. The performance of an error-correcting code is evaluated by measuring both its code rate and its (block or symbol) error rate. Most commonly, there is a trade-offs between them.

The aim of error-correcting codes is designing both a practical encoder and a practical decoder by which low error rate and high code rate are achieved on considering the trade-off.

In 1948, Shannon proved that any channel can be characterized by its *capacity* in the sense that there exists a code so that an information sequence can be decoded with arbitrarily small error rate if and only if the code rate is less than the capacity of the channel [29]. However, the proof does not specify the encoding method except when the code rate is

much less than the capacity. The capacity is called the Shannon limit, which implies that a theoretical limit on the code rate to ensure reliable data transmission.

On the other hand, the designing of practical error-correcting codes was initiated with the development of the Hamming codes in 1950 [12], since then, most studies have been focused on encoding and decoding methods based on the algebraic theory till the mid '90s. In practical systems, codes which are defined by a linear mapping over F_q are often used, where $\mathbb{F}_q$ is a finite field of the order q. Such codes are called *linear codes*. Linear codes is defined on a linear subspace of the vector space $(\mathbb{F}_q)^N$, where N denotes the length of the codeword. Considering a distance over $(\mathbb{F}_q)^N$, an aim of designing encoder is increasing the distance between pairs of codewords as much as possible.

When we design an encoder, we must also design a decoder which is capable of decoding the code defined by the encoder. Considering the error-correcting procedure is represented by a probabilistic model, we can design a decoder which selects the estimate of each information symbols to maximize its posterior probability given a received sequence. This decoding is called the maximum a posteriori probability (MAP) decoding. The MAP decoding can minimize the symbol error rate for any code; that is, it is a best decoder. However, the complexity is exponential in the length of the information sequence except for a class of codes. The class of codes, e.g. convolutional codes, has many pairs of codewords with short distance and then is not good code.

As just described, when we design an error-correcting code, we cannot separate the design of an encoder, which is the code itself, from the design of a decoder.

1.3 Error-correcting Codes on Graphs

In recent years, a new class of codes has been widely studied; the code is represented by a graph and decoded by an algorithm defined on the graph. Turbo codes and Low-Density Parity-Check (LDPC) codes are famous for such codes.

In 1993, Berrou et. al. proposed turbo codes [4]. In the mid '90s, MacKay and Neal [20] and Sipser and Spielman [31] independently rediscovered Low-Density Parity-Check (LDPC) codes [*1]. These codes attracted considerable attention because they can came closest to approaching the Shannon limit with long codeword length. However, a theoretical explanation for the superior performance of their decoding algorithm has not been elucidated.

According to later studies, it is shown that these decoding algorithms are closely related to Pearl's algorithm, which is well known in the field of artificial intelligence [19]. In coding theory, Pearl's algorithm is called the *sum-product algorithm*. The sum-product algorithm is defined on a graph representing a probabilistic model. The graph is called a factor graph. If the factor graph is tree structured, the sum-product algorithm can efficiently compute posterior probabilities of random variables given the specific values of several random variables of the probabilistic model. However, the theoretical explanation of the sum-product

[*1] LDPC codes were first reported by Gallager in 1962 in his Ph.D. thesis [9]. However, the superior applications of the LDPC codes could not be fully demonstrated because of the limited computational resources available in the 1960s, and hence they were not extensively studied in coding theory.

algorithm has not been elucidated in the case where the factor graph is not tree structured.

A factor graph is a bipartite graph consisting of two types of nodes: a variable node and a factor node. Variable nodes represent the random variables in a probabilistic model, and factor nodes represent local constraints among the random variables. A factor node is connected to the variable nodes via undirected edges such that the local constraints of the factor node relate to the random variables corresponding to variable nodes, as shown in Fig. 1.3. In the case of error-correcting codes, variable nodes represent information symbols, codeword symbols, receiver symbols, and other variables if any. Factor nodes represent local constraints among codeword symbols of the code and a channel model. If the factor graph representing a code is tree structured, we can compute posterior probabilities of each information symbols given a received sequence. Then we can apply the MAP decoding to the code.

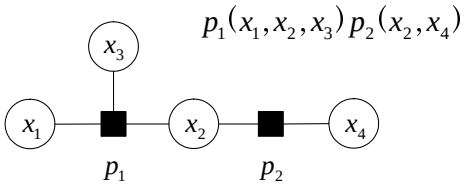

Fig. 1.3 Example of a factor graph. The circular nodes denote the variable node, and the square nodes denote the factor nodes.

The factor graphs representing turbo codes and LDPC codes are not tree structured. Then we cannot perform the MAP decoding by the sum-product algorithm to these codes [*2]. If we perform the sum-product algorithm over the non-tree structured factor graph, we can obtain *pseudo posterior probabilities* of each information symbols. Although they are not exact posterior probabilities, we can perform the *pseudo* MAP decoding that treats the pseudo posterior probabilities as the exact ones. It is shown that good decoding performance can be obtained by the pseudo MAP decoding for turbo codes and LDPC codes. Moreover, the complexity of the decoding is polynomial in the length of the information sequence. We call the decoding methods using the sum-product algorithm is called the *sum-product decoding* whether or not the factor graph is tree structure.

Turbo codes and LDPC codes are defined as not a specific code but an *ensemble* of codes. Then, many researchers proposed code design methods for these codes to improve the performance using the sum-product decoding.

1.4 Graph Representations for Codes

The factor graph not only represents both the code and the channel but also defines the procedure of decoding. When we design a code using the sum-product decoding on a channel,

[*2] As stated below, any factor graph can be transformed to tree structured one. However, the sum-product algorithm cannot efficiently compute posterior probabilities over such the transformed tree structured factor graph.

we must consider not only the performance of the code itself but also the performance of the sum-product decoding over the graph representing both the code and the channel.

For turbo codes and LDPC codes, it is shown that they have a good distance relation between pairs of codewords [20] [23]. In addition, they show good performance using the sum-product decoding on commonly used channels and the complexity of the decoding is low. Therefore, we say they not only are good codes themselves but also have an appropriate graph representation for the sum-product decoding. In this thesis, we focus our discussion on their graph representations of turbo codes and LDPC codes and propose new encoding and decoding schemes based on graph representations.

First, we present characteristics of the graph representations of turbo codes and LDPC codes.

1.4.1 Factor Graph Representing Turbo Codes

A turbo codes is the parallel concatenated codes of two codes which is called convolutional codes. The information sequence is distinctly encoded by two convolutional codes. Both the factor graphs representing the two codes have the variable nodes representing information symbols. Therefore, the concatenated code can be represented by the merged factor graph of the two factor graphs by connecting the variable nodes representing information symbols, as shown in Fig. 1.4. The sum-product decoding of turbo codes uses the merged graphs.

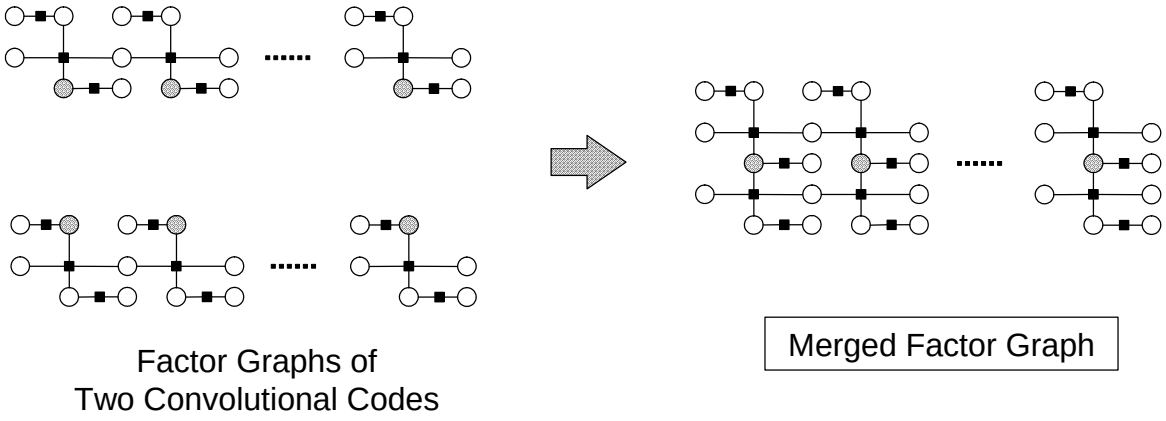

Fig. 1.4 Factor graph representing a turbo code. The gray variable nodes denote the information symbols.

The encoder of turbo codes has a module called an *interleaver*, which permutes the order of symbols in an information sequence. While one convolutional encoder encodes the original information sequence, the other convolutional encoder encodes the permuted information sequence by the interleaver, as shown in Fig. 1.5. It is interesting to note that the design of the permutation pattern depends the performance of turbo codes [23].

After encoding, each output is punctured; puncturing is the process of removing some codeword symbols from the codeword sequence. Puncturing is used for rate control of codes; the puncturing pattern depends on the decoding performance.

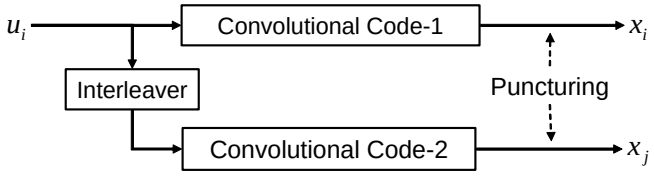

Fig. 1.5 Encoder for turbo codes.

We focus on these characteristics of turbo codes and propose an adaptive encoding scheme using a feedback channel.

1.4.2 Factor Graph Representing LDPC Codes

The factor graph representing an LDPC code is shown as Fig. 1.6, omitting the subgraph representing the channel. All variable nodes denote the codeword symbols in the figure. A factor node representing each local constraint of the code connects variable nodes corresponding codeword symbols that relate the constraint.

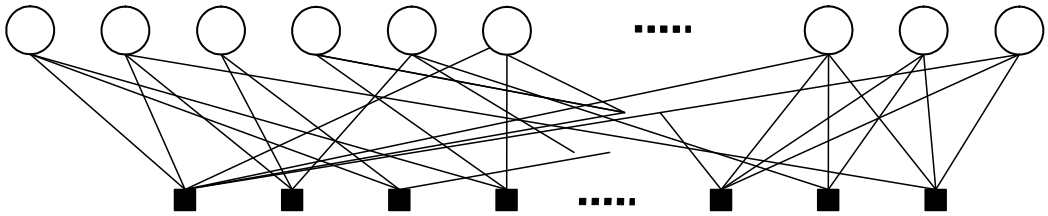

Fig. 1.6 Factor graph representing an LDPC code.

Many studies have reported that the empirical performance of LDPC codes using the sum-product decoding depends on the length of the cycles in the factor graph. It was demonstrated that cycles having small lengths, particularly those with a length of 4 as shown in Fig. 1.7, have a negative effect on the decoding performance [20]. To overcome this negative effect, the use of LDPC codes, which do not have small cycles has been proposed [13] [15]. It should be noted that this approach involves the designing of codes and thus would introduce restrictions on the design of the codes. On the other hand, we can solve this problem by transformations of a factor graph without changing the code itself.

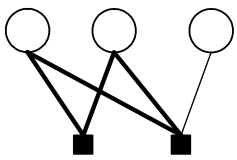

Fig. 1.7 Cycle with a length of 4.

1.5 Design of Encoding and Decoding Schemes Using Graph Representations

This thesis describes two new schemes taking into consideration graph representations. One is developing for a scheme of an adaptive encoding scheme using a feedback channel by focusing on the characteristics of parallel concatenated codes. The other is a scheme for the improvement of the performance of LDPC codes with small cycles by transformations of a factor graph.

1.5.1 An Adaptive Encoding Scheme Using a Feedback Channel

In Chapter 3, we propose an adaptive encoding scheme using a feedback channel. A feedback channel is a channel in which the transmitter can receive information from the receiver. J. M. Shea proposed the following procedure using the feedback channel in [30]. After the transmitter transmits a codeword, the receiver provides *unreliable symbol positions* of the information sequence to the transmitter. An unreliable symbol position is the symbol position whose value cannot be easily determined by the sum-product decoding. The transmitter then sends no encoded information symbols which correspond to the unreliable symbol positions.

We extend this procedure by focusing on the characteristics of parallel concatenated codes and propose an adaptive encoding scheme using a feedback channel.

We consider the following encoding scheme. The transmitter transmits a codeword. After the receiver provides the unreliable symbol positions to the transmitter, the transmitter adaptively encodes the information sequence based on the unreliable symbol positions and transmits the codeword. This procedure can be repeated until a termination condition is satisfied. The receiver can perform the sum-product decoding using both the merged factor graph representing the all codes and all received sequences.

In this extended scheme, the additional codes must be able to achieve higher correcting capabilities for the unreliable symbol positions than the other symbol positions in the information sequence. In addition, it must be able to do adaptively based on unreliable symbol positions.

We propose such a additional code based on the characteristics of turbo codes. We use convolutional codes; these codes are designed similar as a turbo code as shown in Fig. 1.8. After the receiver provides the unreliable symbol positions to the transmitter, the transmitter adaptively designs both the permutation pattern of the interleaver and puncturing pattern on the basis of the unreliable symbol positions. Then, the permutated information sequence is encoded by the additional code, and the codeword is transmitted.

Since only the permutation and puncturing patterns are dynamically changed, the complexity introduced by the adaptive feature is low. We will describe design methods for the permutation pattern of the interleaver and the puncturing pattern, by which a high error-correcting capability for unreliable symbol positions is achieved. We demonstrate the

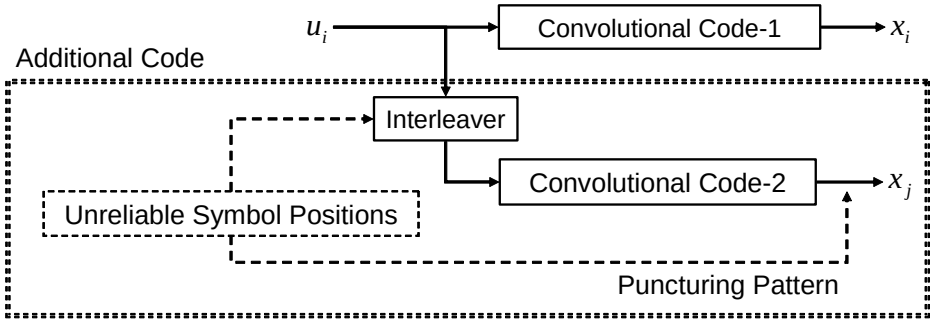

Fig. 1.8 Proposed adaptive encoding method.

performance of our proposed method by carrying out numerical experiments.

1.5.2 Transformations of a Factor Graph for LDPC Codes

In Chapter 4 and Chapter 5, we focus on transformations of a factor graph. A code can be represented by several factor graph representations; that is, a code and a factor graph is not a one-to-one relation. Several transform methods of factor graphs have been proposed. By transforming the factor graph, we can improve the performance of the sum-product decoding without changing the code itself.

In Chapter 4, we deal with the *clustering method*, which was proposed by J. Pearl in the study of artificial intelligence [24] for a graph called the Bayesian network. We can apply this concept to factor graphs.

By the clustering method, a graph can be transformed by introducing a random variable which is defined by the direct product of several random variables; it yields that several variable nodes are combined into a single variable node on the graph, as shown in Fig 1.9. This results in an increase in the complexity of the sum-product decoding, since the complexity of the sum-product algorithm is proportional to the cardinality of the domain of a random variable. If a non tree-structured graph can be transform to a tree-structured one by the clustering method, we can compute the exact posterior probabilities by the sum-product algorithm.

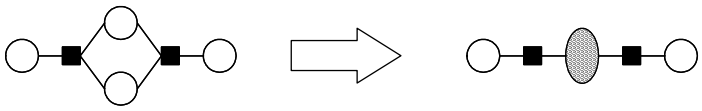

Fig. 1.9 Simple example of the clustering method.

In the case of LDPC codes, to obtain a tree-structured factor graph, almost all nodes in the factor graph need to be combined. This results in exponential increase in the complexity of the sum-product decoding. It is not practical.

Therefore, we propose a method in which only the subgraphs corresponding to small cycles are transformed by the clustering method. Our proposed method implies that the

accuracy of the computation of approximate posterior probabilities increases by increasing the complexity of the sum-product algorithm of the subgraph.

By carrying out numerical experiments, we have verified that our method can improve the performance of the sum-product decoding with LDPC codes having small cycles; in other words, the degradation of the decoding performance due to small cycles is successfully mitigated by our method without changing the code itself.

In Chapter 5, we deal with a transform method for factor graphs representing LDPC codes on a binary erasure channel (BEC). The sum-product algorithm on the BEC can be simplified to a message-passing decoding. The performance of the message-passing decoding is determined by certain combinatorial structures of the factor graph, which is known as *stopping sets* [6]. Stopping sets are defined not on a code but on a factor graph. Therefore, we can improve the performance of the message-passing decoding by transforming the factor graph.

We propose a new transform method of a factor graph to improve the performance of the message-passing decoding. By using the characteristics of the stopping sets, we show conditions of channel erasure patterns which can be corrected by performing the message-passing decoding over the transformed factor graph but cannot be corrected by performing the message-passing decoding over the original factor graph.

Moreover, we show a relation between the transform method and the subgraphs corresponding to small cycles. By carrying out numerical experiments, we have verified that the transform method can improve the performance of the message-passing decoding with LDPC codes having small cycles.

Chapter 2

Preliminary

In this chapter, we describes a brief overview of error-correcting codes. As for details, refer to the literatures by S. Lin and D. J. Costello [17], and T. Richardson and R. Urbanke [26] etc.

We use the following notation and terminology throughout this thesis. The word *sequence* means a row vector of symbols. Let $|\mathbf{a}|$ and $|\mathbf{S}|$ be the number of elements of a sequence $\mathbf{a}$ and set $\mathbf{S}$, respectively. Let $\mathbf{Z}^{\mathbf{T}}$ be the transposed matrix of a matrix $\mathbf{Z}$, and $\mathbf{a}^{\mathbf{T}}$ be the transposed vector of a vector $\mathbf{a}$.

2.1 Error-correcting Codes

2.1.1 Basic Definitions and Notations for Error-correcting Codes

In this thesis, we consider the following error-correcting procedure, as shown in Fig. 2.1.

An information source emits a digital signal called an *information symbol*. Let us consider a sequence of information symbols with a finite length, called an *information sequence*, which is denoted by $\mathbf{u}$. Each information symbol is denoted by u_i, where the subscript means the symbol position in $\mathbf{u}$. Let K denotes the length of the information sequence. The information symbol is represented by a member of the finite set $\mathscr{U}$. In this study, let

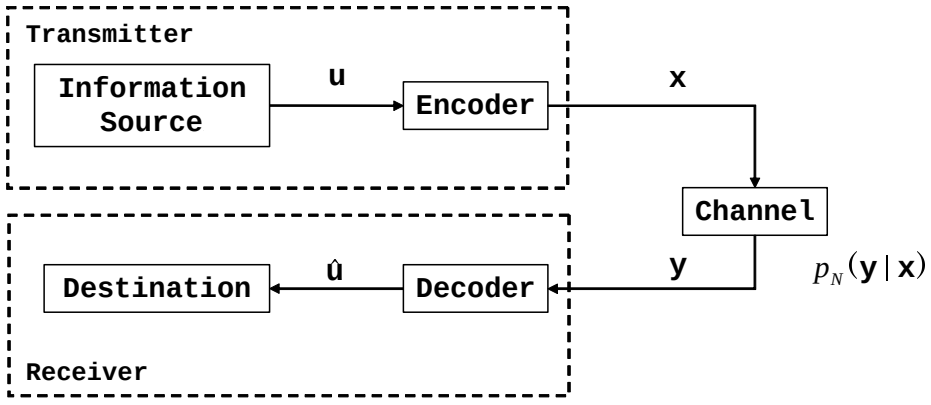

Fig. 2.1 Error correction.

$\mathscr{U} = \{0, 1\}$, as in a conventional case. Then, we have $\mathbf{u} = u_1 u_2 ... u_K \in \{0, 1\}^K$.

Let us consider a mapping from the information sequence to a *codeword*, which is denoted by $\mathbf{x}$. That is, define a mapping from $\{0, 1\}^K$ to $\mathscr{X}^N$, where $\mathscr{X}$ denotes a finite set with q elements and N denotes the length of the codeword. Such a mapping is referred to as an *encoder*. The range is called a *code*, which is denoted by C, and the sequences in C is called the codewords. The symbol of the codeword is called a *codeword symbol*, which is denote by x_i, where the subscript means the symbol position in $\mathbf{x}$. In this thesis, we deal with an injection mapping; that is, there is a one-to-one correspondence between a information sequence and its corresponding codeword. We consider only binary codewords; that is, let $\mathscr{X} = \{0, 1\}$. Then, we have $\mathbf{x} = x_1 x_2 ... x_N \in \{0, 1\}^N$.

A code in which all information symbols are embedded in a codeword is called *systematic codes*; that is K specified symbol positions of the codeword coincide the corresponding information symbols.

The transmitter encodes the information sequence u to the codeword x by using the encoder. Once the transmitter transmits the codeword $\mathbf{x}$ through a channel, the receiver receives a noise-corrupted codeword, which is called a *received sequence*. The received sequence is denoted by $\mathbf{y}$, and its length is N. Each symbol of the received sequence is called a *received symbol*, which is denoted by y_i. The received symbol is denoted by a member of the finite set $\mathscr{Y}$. Then, we have $\mathbf{y} = y_1 y_2 ... y_N \in \mathscr{Y}^N$.

The receiver estimates the transmitted information sequence by using the given received sequence. Therefore, let us consider a mapping from $\mathscr{Y}^N$ to $\{0, 1\}^K$. This mapping is called a *decoder*. The estimator of the information sequence is denoted by $\hat{\mathbf{u}}$.

2.1.2 Probabilistic Model for Error-correcting Codes

Assume that the information symbols occur according to a probability distribution, and assume that the channel is governed by a probability distribution of an output sequence given an input sequence. Then, the above error-correcting procedure is represented by a probabilistic model of which each symbol is represented by a random variable.

We define random variables corresponding to each sequence and each symbol. Let $\mathbf{U}$ be the random variable representing the information sequence $\mathbf{u}$, and let U_i be the random variable representing the information symbol u_i. Let $\mathbf{X}$ be the random variable representing the codeword $\mathbf{x}$, and let X_i be the random variable representing the codeword symbol x_i. Let $\mathbf{Y}$ be the random variable representing the received sequence $\mathbf{y}$, and let Y_i be the random variable representing the received symbol y_i.

Assuming that the source coding problem is solved, let an information symbol be uniform independent identically distributed (i.i.d.) symbols which are equally likely to be zero or one. That is $\Pr\{U_i = 0\} = \Pr\{U_i = 1\} = 1/2$ for all $i (1 \leq i \leq K)$.

We define a channel by a probability distribution $p_N(\mathbf{y}|\mathbf{x})$.

2.1.3 Performance Measures for Error-correcting Codes

Code rate is defined as $R = \frac{\log_q |C|}{N}$, which represents the redundancy due to the encoding process. In the case of using the binary information sequence and binary codeword, we have $R = \frac{K}{N}$.

A block error is said to occur when $\mathbf{u} \neq \hat{\mathbf{u}}$, and the probability of the occurrence of the block error is called *block error rate*. Similarly, a symbol error is said to occur when $u_i \neq \hat{u}_i$. The probability of the occurrence of the symbol error is called *symbol error rate*. The performance of an error-correcting code is evaluated by measuring both its code rate and its (block or symbol) error rate. Most commonly, there is a trade-offs between them.

The aim of error-correcting codes is designing both the encoder and the decoder by which low error rate and high code rate are achieved on considering the trade-off. For practical use, both the encoder and the decoder should be able to be performed with low complexity.

2.2 Channel Models

In this thesis, we consider only the class of stationary memoryless channels; noise occurs for each time point independently and stationary. That is, the channel model is represented by $p_N(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{N} p(y_i|x_i)$, where $p(y_i|x_i)$ is defined as a conditional probability distribution of the received symbol y_i given the codeword symbol x_i.

In this thesis, we consider two stationary memoryless channel models; Binary-Input Additive White Gaussian Noise (BIAWGN) channel and Binary Erasure Channel (BEC).

2.2.1 Binary-Input Additive White Gaussian Noise (BIAWGN) Channel

The BIAWGN channel has two discrete input alphabets and continuous output alphabets. Each codeword symbol is transformed by the binary-bipolar conversion; that is, each $\{0, 1\}$ symbol is converted to $\{+1, -1\}$. The converted signal is denoted by c_i. The received symbol y_i takes on the real values; that is, $\mathscr{Y} \in \mathbb{R}$. Let n_i be a Gaussian random process with zero mean, and its variance is σ^2. The received symbol is denoted by the addition of c_i and n_i over real numbers; that is,

$$y_i = c_i + n_i. \tag{2.1}$$

The conditional probability distribution $p(y_i|x_i)$ of the BIAWGN channel is denoted by

$$p(y_i|0) = \frac{1}{\sqrt{\pi\sigma^2}} \exp\left(-\frac{(y_i - 1)^2}{2\sigma^2}\right), \tag{2.2}$$

$$p(y_i|1) = \frac{1}{\sqrt{\pi\sigma^2}} \exp\left(-\frac{(y_i + 1)^2}{2\sigma^2}\right). \tag{2.3}$$

We have $\sigma^2 = N_0/2$, where N_0 is one-sided power spectral density of the process. Let E_b denotes energy per an information symbol. Then, E_b/N_0 means the energy per symbol to noise power spectral density ratio and is an important parameter of the BIAWGN channel.

2.2.2 Binary Erasure Channel (BEC)

The BEC introduced by Elias as a toy example in 1954, is the simplest but non-trivial channel model. The received symbol y_i takes on values in the alphabet $\{0, 1, ?\}$, where the character ? indicates an *erasure*. Each transmitted symbol is either erased with probability ϵ, or received correctly. Figure 2.2 is shown the relation between input symbols and output symbols of the BEC with parameter ϵ.

The conditional probability distribution $p(y_i|x_i)$ of the BEC is denoted by

$$p(0|0) = p(1|1) = 1 - \epsilon, \tag{2.4}$$

$$p(0|1) = p(1|0) = 0, \tag{2.5}$$

$$p(?|0) = p(?|1) = \epsilon. \tag{2.6}$$

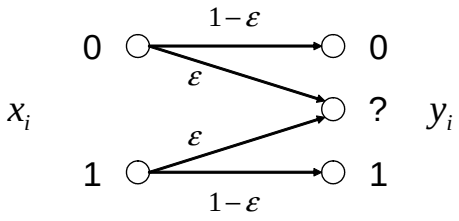

Fig. 2.2 Binary erasure channel with parameter ϵ.

2.3 Linear Codes

In this section, we describe *linear codes*. Linear codes allow for efficient encoding and decoding algorithms, and then are used in many error-correcting systems.

When a code is defined as a linear subspace of the vector space of the finite field, the code is called a *linear code*. Let $\mathbb{F}_2$ denotes the binary field consisting of $\{0, 1\}$ with mod-2 addition and mod-2 multiplication. That is, a code C over a field $\mathbb{F}_2$ is linear if it is closed under N-tuple addition and scalar multiplication:

$$k\mathbf{x} + k'\mathbf{x}' \in C \quad \forall\, \mathbf{x}, \mathbf{x}' \in C \text{ and } k, k' \in \mathbb{F}_2. \tag{2.7}$$

There are two structurally different type of linear codes: linear block codes and convolutional codes.

2.3.1 Linear Block Codes

Linear block codes are linear codes such that an information sequence is encoded independently of the other information sequences. Since a linear block code C of length N over $\mathbb{F}_2$ is a subset of $\mathbb{F}_2^N$, there must exist an integer K so that C has a dimension K. It is customary to call a $K \times N$ matrix G, whose rows form a linearly independent basis for C, a

generator matrix for C. Conversely, given a matrix $G \in \mathbb{F}_2^{K\times N}$ of rank K we can associate with it the code $C(G)$:

$$C(G) = \{\mathbf{x} \in \mathbb{F}_2^N : \mathbf{x} = \mathbf{u}G, \mathbf{u} \in \mathbb{F}_2^K\} \tag{2.8}$$

To each linear code C, we associate the dual code $C^\perp$:

$$C^\perp = \{\mathbf{v} \in \mathbb{F}_2^N : \mathbf{x}\mathbf{v}^T = 0, \forall \mathbf{x} \in C\} = \{\mathbf{v} \in \mathbb{F}_2^N : G\mathbf{v}^T = 0^T\}. \tag{2.9}$$

Assume that $\mathbf{v}$ and $\mathbf{v}'$ are elements of $C^\perp$ and that $\mathbf{x}$ is an element of C. Since $\mathbf{x}\mathbf{v}^T = 0 = \mathbf{x}(\mathbf{v}')^T$ implies $x(\alpha\mathbf{v} - \mathbf{v}')^T = 0$ for any $\alpha \in \mathbb{F}_2$, it follows that $C^\perp$ is a linear code as well. Therefore, it has a basis. It is customary to denote such a basis by H. It is also said to be a *parity-check matrix* of the original code C. We have

$$C = \{\mathbf{x} \in \mathbb{F}_2^N : \mathbf{x} = \mathbf{u}G, \mathbf{u} \in \mathbb{F}_2^K\} = \{\mathbf{x} \in \mathbb{F}_2^N : H\mathbf{x}^T = 0^T\} \tag{2.10}$$

Let N and M be the number of columns and rows of H, respectively. Let H_{mn} be the value of m-th rows and n-th columns of H $(1 \le m \le M)$. We define sets which indicate the positions of the symbol one in H: $N(m) := \{n|H_{mn} = 1\}$ and $M(n) := \{m|H_{mn} = 1\}$.

The m-th row of H means that $\sum_{k\in N(m)} x_k = 0 \mod 2$. The constraint is called a *parity-check* constraint.

2.3.2 Convolutional Codes

Convolutional codes are linear codes so that the encoding operation can be viewed as a convolution. The encoder for a convolutional code accepts k-symbol block of the information symbols and produces n-symbol blocks of the codeword symbols. However, each encoded block depends not only the corresponding a block of the information symbols at the same time unit but also on r-previous blocks of the information symbols. We say the encoder has a memory order of r. Then, the code rate is denoted by k/n. However, since a termination process, as stated below, is commonly performed for improvement of the performance of decoding, the code rate is decreased a little from k/n.

The encoder for a convolutional code is implemented with a sequential logic unit. Figure 2.3 shows an example of an encoder of a convolutional code with $k = 1$, $n = 2$, and $r = 3$. Because this encoder has no feedback lines in the circuit, this is called a *feed-forward convolutional* code. In this Figure, the square symbol denotes a delay element, which delays a symbol by one clock time for each stage. The circular with the symbol '+' denotes the computation rule of addition modulo 2. We decide the location of the position of codeword symbol with applying the rule that the encoder takes the first encoded block from the top of output line. For example, $\mathbf{u} = 10110$ is encoded to $\mathbf{x} = 1001111000$ by this convolutional encoder.

A convolutional code can be defined by representation of a generator matrix. The generator matrix of the above feed-forward convolutional code is denoted by

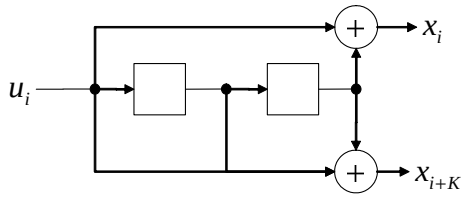

Fig. 2.3 Encoder of a feed-forward convolutional code.

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 & & & & & \\ & & 1 & 1 & 0 & 1 & 1 & 1 & & & \\ & & & & 1 & 1 & 0 & 1 & 1 & 1 & \\ & & & & & & 1 & 1 & 0 & 1 & \\ & & & & & & & & 1 & 1 & \dots \\ & & & & & \vdots & & & & & \end{bmatrix}. \tag{2.11}$$

It is customary to denote a generator matrix of a convolutional code to a polynomial of a *delay operator* D. This is called the *generator polynomial* of a convolutional code.

The generator polynomial for the top of output line of the above feed-forward convolutional code is denoted by $g^1(D) = 1 + D^2$. The generator polynomial for the bottom of output line is denoted by $g^2(D) = 1 + D + D^2$. We denote the generator polynomials as the following matrix:

$$G(D) = \begin{bmatrix} 1 + D^2 & 1 + D + D^2 \end{bmatrix}. \tag{2.12}$$

In general, this matrix has n columns and k rows, and each element is a r-ordered polynomial of D.

A convolutional code with feedback lines in its circuit is called a *feedback convolutional code*, as shown in Fig. 2.4. The generator polynomial of the feedback convolutional code can be represented with ratios of polynomials. The generator polynomial the above feedback convolutional code is denoted by

$$G(D) = \begin{bmatrix} 1 & \dfrac{1 + D^2}{1 + D + D^2} \end{bmatrix}. \tag{2.13}$$

When a feedback convolutional code is systematic, the code is called a recursive systematic convolutional (RSC) code. The above feedback convolutional code is an RSC codes. In general, the range of the weight of the codewords of RSC codes corresponding to information sequences which have a same weight is larger than that of feed-forward convolutional codes. This is an important characteristic of RSC codes.

We can interpret the encoding of convolutional codes as a finite state machine with a state transition diagram by representing the values in delay elements as internal state of the finite state machine. For example, the above feedback convolutional code can be represented by a finite state machine with the state transition diagram as shown in Fig. 2.5. S_{ab} denotes the state corresponding that two delay elements takes the values a and b, respectively. The numbers of edge n_i/n_o mean that the sequence n_i is encode to the sequence n_o.

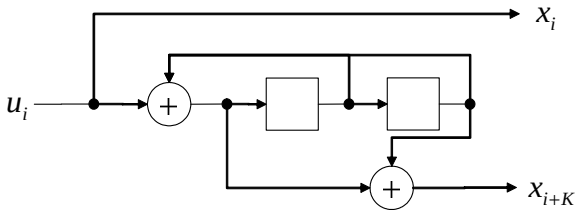

Fig. 2.4 A systematic feedback convolutional encoder.

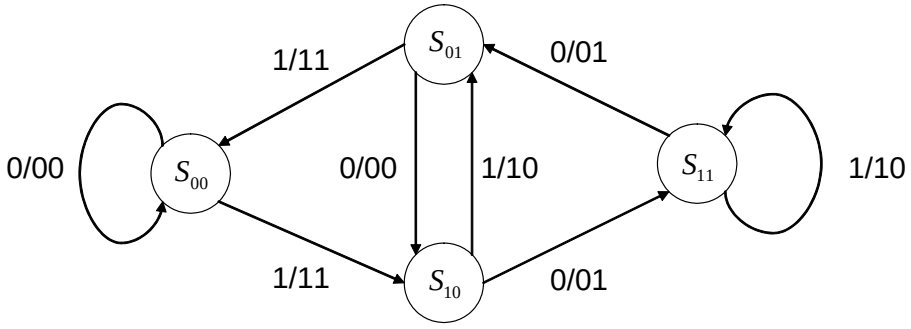

Fig. 2.5 The state transition diagram of a feedback convolutional code.

The state of convolutional codes is commonly initialized to be the all zero state; that is, all delay elements are initialized to the value zero. It is convenient to finalize the state to be the all zero state for commonly used decoding algorithms, e.g. the BCJR decoding as stated below. For this purpose, some extra symbols are input to transmit the state to the all zero state after all blocks of information symbols are encoded. The process is called a termination process, which can improve the performance of the decoding algorithms.

2.3.3 Hamming Weight and Hamming Distance

It is customary to define the distance relation between pairs of codewords. Hamming distance and Hamming weight involve the error-correcting capability of codes. We may call them weight and distance for short, respectively.

Definition 2.1 (Hamming weight and Hamming distance)**.** *Let* $\mathbf{u}, \mathbf{v} \in \mathbb{F}_2^N$*. The Hamming weight of a sequence* $\mathbf{u}$ *is equal to the number of non-zero symbols in* $\mathbf{u}$*. The Hamming distance of a pair* $\mathbf{u}, \mathbf{v}$ *is the number of positions in which* $\mathbf{u}$ *differs from* $\mathbf{v}$*.*

Roughly speaking, codewords kept far apart in the vector space $(\mathbb{F}_2)^N$ at Hamming distance, so that when a noise-corrupted codeword is received it is still possible to determine which codeword was transmitted. Therefore, a pair of codewords with small distance has a negative effect of the error-correcting capability, particularly when the noise of the channel is moderate. An aim of designing codes is increasing the distance between pairs of codewords as much as possible.

Choosing $k = 1$ and $\mathbf{x} = \mathbf{x}'$ for Eq. (2.7) shows that the all-zero sequence is a code-

word of any linear code. Then the weight of codewords becomes an important measure of linear codes. In addition, the *minimum weight* of a linear code affects the error-correcting capability, particularly when the noise of the channel is moderate.

2.4 Code Constraints and Their Factorization

It is convenient to define the following indicator function of a code C representing its *code constraint*:

$$\Psi_C(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in C; \\ 0 & otherwise. \end{cases} \tag{2.14}$$

If the code is a linear code, the indicator function can be factorized.

For linear block codes, the indicator function is factorized using their parity-check constraints as following:

$$\Psi_C(\mathbf{x}) = \prod_{m=1}^{M} \psi_m(\{x\}_m), \tag{2.15}$$

where let $\{x\}_m$ be the set of codeword symbols such that $\{x_i | i \in N(m)\}$, and

$$\psi_m(\{x\}_m) = \begin{cases} 1 & \sum_{k \in N(m)} x_k = 0 \mod 2; \\ 0 & otherwise. \end{cases} \tag{2.16}$$

For a convolutional code, the indicator function is factorized introducing variables denoting the state of the encoder. We call the variables *state variables*. Let $\mathscr{S}$ be the domain of the state variable, and then we have $|\mathscr{S}| = 2^r$. Let $s_i \in \mathscr{S}$ denotes the state variable corresponding to the input information symbol u_i, and let s_0 denotes the initial state. We denote the sequence of state variables by $\mathbf{s} = s_0, s_1, ...s_K$.

Here for simplicity of notation, we describe only systematic feedback convolutional codes with $R = 1/2$ such that an information symbol u_i is encoded to the pair of a codeword symbol $x_i (= u_i)$ and x_{i+K}. In the case, the indicator function with introducing the state variable is factorized as:

$$\Psi_C(\mathbf{x}, \mathbf{s}) = \prod_{i=1}^{K} \delta(s_{i-1}, s_i, x_i, x_{i+K}), \tag{2.17}$$

where let $\delta(s_{i-1}, s_i, x_i, x_{i+K})$ is a function that takes the value 1 if the state s_{i-1} can transition to the state s_i associated with the input $x_i (= u_i)$ and the output x_{i+K}, and takes the value 0 if not.

2.5 Maximum Likelihood (ML) and the Maximum a Posteriori Probability (MAP) Decoding

Because the error-correcting procedure is represented by a probabilistic model, we can consider the following maximum likelihood (ML) decoding and maximum a posteriori probability (MAP) decoding.

One estimation method of the information sequence given a received sequence $\mathbf{y}$ is to maximize its posterior probability given a received sequence; that is

$$\hat{\mathbf{u}} = \underset{\mathbf{u}\in\{0,1\}^K}{\operatorname{argmax}} \Pr\{\mathbf{U}=\mathbf{u}|\mathbf{Y}=\mathbf{y}\}. \tag{2.18}$$

From the assumption that information symbols are uniform i.i.d. symbols which are equally likely to be zero or one, the posterior probability equals the likelihood:

$$\hat{\mathbf{u}} = \underset{\mathbf{u}\in\{0,1\}^K}{\operatorname{argmax}} \Pr\{\mathbf{Y}=\mathbf{y}|\mathbf{U}=\mathbf{u}\}. \tag{2.19}$$

The decoding following this estimation method is called the ML decoding.

The other estimation method is symbol-wise estimation; that is, we select the estimate of each information symbols to maximize its posterior probability given a received sequence:

$$\hat{u_i} = \underset{u_i\in\{0,1\}}{\operatorname{argmax}} \Pr\{U_i=u_i|\mathbf{Y}=\mathbf{y}\}. \tag{2.20}$$

$\Pr\{U_i = u_i|\mathbf{Y}=\mathbf{y}\}$ is obtained by marginalization of the joint posterior probability $\Pr\{\mathbf{U}=\mathbf{u}|\mathbf{Y}=\mathbf{y}\}$ over information symbols other than u_i; that is,

$$\hat{u_i} = \underset{c\in\{0,1\}}{\operatorname{argmax}} \sum_{\{\mathbf{u}\in\{0,1\}^K|u_i=c\}} \Pr\{\mathbf{U}=\mathbf{u}|\mathbf{Y}=\mathbf{y}\}. \tag{2.21}$$

We call this estimation the maximum a posteriori probability (MAP) decoding, as in a conventional case, and call the marginal posterior probability $\Pr\{U_i=u_i|\mathbf{Y}=\mathbf{y}\}$ a posterior probability, for short.

The ML decoding can minimize the block-wise error probability for any code, while the MAP decoding can minimize the symbol error rate for any code. However, the complexity is exponential in the length of the information sequence; the number of max operations is 2^K in Eq. (2.19), and the number of summations and max operations in Eq. (2.21) is also 2^K. Thus the ML and MAP decoding are not practical.

In the next section, we will describe an algorithm to be able to efficiently compute the posterior probabilities under a condition. The algorithm uses the factorization of a function to efficiently compute them. Then, we first consider to factorize the right term of Eq. (2.21):

$$\sum_{\{\mathbf{u}\in\{0,1\}^K|u_i=c\}} \Pr\{\mathbf{U}=\mathbf{u}|\mathbf{Y}=\mathbf{y}\} = \sum_{\{\mathbf{x}\in\{0,1\}^N|u_i=c\}} \Pr\{\mathbf{X}=\mathbf{x}|\mathbf{Y}=\mathbf{y}\} \tag{2.22}$$

$$= \sum_{\{\mathbf{x}\in\{0,1\}^N|u_i=c\}} \frac{p_N(\mathbf{y}|\mathbf{x})\Pr\{\mathbf{X}=\mathbf{x}\}}{\Pr\{\mathbf{Y}=\mathbf{y}\}} \tag{2.23}$$

$$= \sum_{\{\mathbf{x}\in\{0,1\}^N|u_i=c\}} \alpha\Psi_C(\mathbf{x})\prod_{i=1}^{N} p(y_i|x_i). \tag{2.24}$$

by using the Bayes' theorem and the channel definition, and where let α be a normalized constant. If the code is a linear code, $\Psi_C(x)$ can be also factorized, as stated the last section. If the code is a convolutional code, state variables need to be marginalized.

As just described, the equation denoting the MAP decoding for a linear code can be factorized into two parts of terms: the term denoting the code constraints and the term denoting the channel.

2.6 Factor Graph and the Sum-product Algorithm

This section describes factor graphs and the sum-product algorithm. These play an important role for error-correcting codes on graphs.

2.6.1 Factor Graph

We consider a function of variables $a_1, a_2...$ which can be factorized as following :

$$p(V) = \prod_j f_j(A_j), \tag{2.25}$$

where $V := \{a_1, a_2, ...\}$ and A_j is a subset of V. We call each constituent term of the product f_j a *local function*.

A factor graph is a graphical model representing a factorization of a function. This is a bipartite graph with two types of nodes: a variable node and a factor node. A variable node drawn by a circle represents a variable. A factor node drawn by a square represents a local function. We call the variable node corresponding to a_i "variable node a_i" and the factor node corresponding to f_j "factor node f_j", for simplicity. A variable node is connected by edges to factor nodes when the corresponding variable is constrained by the corresponding local function. The number of edge emitting from a node is called a *degree*.

2.6.2 Computation of Marginal of a Function

The sum-product algorithm is an algorithm which can efficiently compute marginals of a function with respect to all variables. It corresponds to compute marginal posterior probabilities in the case of that the function denotes a probabilistic model and that specific values of several random variables are obtained.

Before we describe the procedure of the sum-product algorithm, we show why it can compute a marginal of a function when a factor graph is tree with a simple example of distributive law, as like in [26].

2.6.3 Distributive Law

Let $\mathbb{F}$ be a field and let $a, b, c \in \mathbb{F}$. By the *distributive law*, we have

$$ab + ac = a(b + c). \tag{2.26}$$

This simple law can significantly reduce computational complexity of the marginal of a

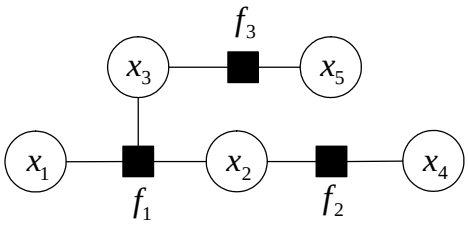

Fig. 2.6 The factor graph representing Eq. (2.27).

function. For example, consider the following function:

$$f(z_1, z_2, z_3, z_4, z_5) = f_1(z_1, z_2, z_3) f_2(z_2, z_4) f_3(z_3, z_5), \tag{2.27}$$

where $z_i \in \mathbb{F}$ for $1 \leq i \leq 5$.

The factor graph representing Eq. (2.27) is shown in Fig. 2.6. We introduce the notation $\sum_{\sim V(z_i)}$ to represent a summation over all variables contained in the expression according to a set of variables V except the z_i. We are interested in computing the marginal of f with respect to z_1, and denote this marginal $f(z_1)$:

$$f(z_1) = \sum_{\sim V(z_1)} f(z_1, z_2, z_3, z_4, z_5). \tag{2.28}$$

Determining $f(z_1)$ for all values of z_1 by brute force requires $O(\mathbb{F}^5)$, where we assume all operations (addition, multiplication, function evaluation, etc.) have the same cost. However, by the distributive law, we can rewrite $f(z_1)$ as

$$f(z_1) = [\sum_{z_2, z_3} f_1(z_1, z_2, z_3)[\sum_{z_4} f_2(z_2, z_4)][\sum_{z_5} f_3(z_3, z_5)]]. \tag{2.29}$$

It is easy to confirm the number of operations are reduced. While the factor graph is tree structured, such a factorization can be done generally. Then, we can compute marginal of any function by using this procedure. The sum-product algorithm is the generalization of this procedure.

2.6.4 Sum-product Algorithm

The sum-product algorithm is a message-passing type algorithm defined on the factor graph. Messages denote variables assigned on each edge of the factor graph. and its value is computed by following processes.

If the factor graph is tree structured, the sum-product algorithm can efficiently compute a marginal of a function with respect to all variables. The sum-product algorithm can apply the factor graph which is not tree structured. However, the theoretical explanation of the sum-product algorithm has not been elucidated in the case.

We describe the procedure of the sum-product algorithm over the function (2.25). We define two types of messages : $q_{aj}(z_a)$ and $r_{jb}(z_b)$. $q_{aj}(z_a)$ denotes the messages from

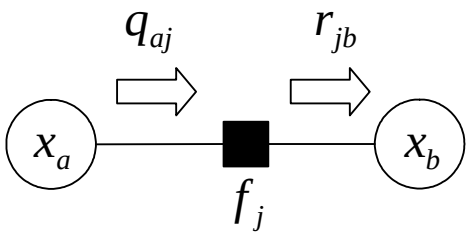

Fig. 2.7 Messages of the sum-product algorithm.

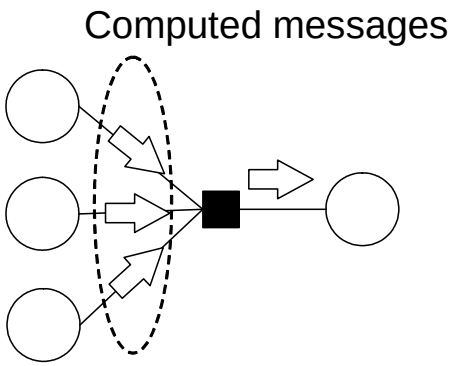

Fig. 2.8 A message computation of the sum-product algorithm.

the variable node z_a to the factor node f_j; the parameter x_a means that each message is distinguish according to the value of z_a. $r_{jb}(z_b)$ denotes the messages from the factor node f_j to the variable node z_b; the parameter z_b means that each message is distinguish according to the value of z_b. These messages are shown in Fig. 2.7.

First, all messages emitted from leaf nodes are initialized. After that messages are computed by alternating two processes: a *variable node process* and *factor node process*. These processes are performed by following an order such that whenever a messages emitted from a node is computed, all incoming messages to the node have already been computed, as like in Fig. 2.8.

We define a set of neighboring factor nodes of a variable node z_i is denoted by $N(i)$, and a set of neighboring variable nodes of a factor node f_j is denoted by $M(j)$. Although these notations are same as the notation about parity-check matrices, they have the similar mean on the factor graph representing codes.

Initialization at leaf nodes

For every message $r_{ji}(z_i)$ that is emitted from a leaf factor node,

$$r_{ji}(z_i) = f_j(z_i). \tag{2.30}$$

For every message $q_{ij}(z_i)$ that is emitted from a leaf variable node,

$$q_{ij}(z_i) = 1. \tag{2.31}$$

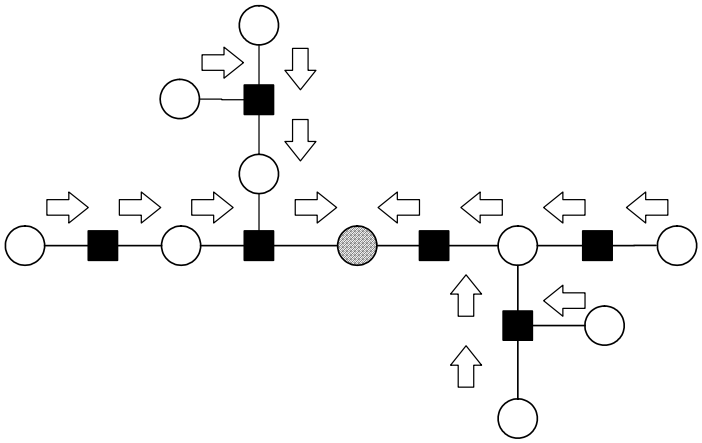

Fig. 2.9 An order of message computation for the gray node when the graph is tree-structured.

Variable Node Process

$$q_{ij}(z_i) = \prod_{m \in M(i) \setminus j} r_{mi}(z_i). \tag{2.32}$$

Factor Node Process

$$r_{ji}(z_i) = \sum_{\sim A_j(z_i)} f_j(A_j) \prod_{n \in N(j) \setminus i} q_{nj}(z_i). \tag{2.33}$$

Marginalization Process

$$\mathrm{app}_i(z_i) = \prod_{j \in M(i)} r_{ji}(z_i) \tag{2.34}$$

2.6.5 Order of Message Computations

When the graph is tree structured, we obtain the marginal a function with respect to any variables by the following specified order of message computations; perform the marginalization processes with respect to the variables after alternating the variable node and factor node processes from all the leaf nodes to the corresponding node as shown in Fig. 2.9.

On the other hand, when the graph is not tree structured, the theoretical explanation of the sum-product algorithm has not been elucidated. Therefore, we cannot specify such a computation order of the processes.

2.6.6 Complexity of the Sum-product Algorithm

The number of multiplications in both Eq. (2.32) and Eq. (2.33) is proportional to the number of edges emitting from a node. The number of additions in Eq. (2.33) is proportional

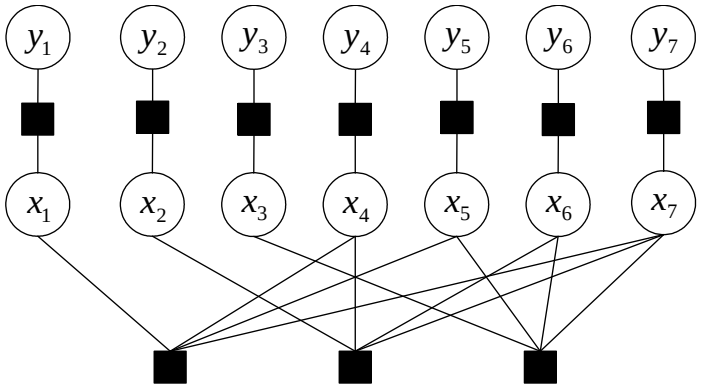

Fig. 2.10 Factor graph representing Eq. (2.35).

to the product of the cardinality of every element of A_j except the a_i. Therefore, if the factor graph is a sparse graph and cardinalities of all variables are small constants which is not proportional to the number of variable N, then the overall complexity of the sum-product algorithm is proportional to the number of executing processes of the variable node and factor node processes.

2.7 Error-correcting Codes on Graphs and the Sum-product Decoding

The equation denoting the MAP decoding for a linear code can be factorized to the form of Eq. (2.25) as stated in Section 2.5. Therefore, we can represent linear codes by factor graphs. First, we show examples of factor graphs representing linear codes.

2.7.1 Factor Graphs Representing Linear Codes

In the case of linear block codes, variable nodes represent information symbols, codeword symbols and receiver symbols. If the code is a systematic code, variable nodes corresponding to information symbols can be omitted. Factor nodes represent parity-check constraints of the code ψ and a channel model. Figure 2.10 shows an example of the factor graph for given the parity-check matrix:

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{2.35}$$

In the case of convolutional codes, variable nodes represent information symbols, codeword symbols, receiver symbols and state variables. Factor nodes represent δ and a channel model. Fig. 2.11 shows the factor graph for systematic feedback convolutional codes with $R = 1/2$.

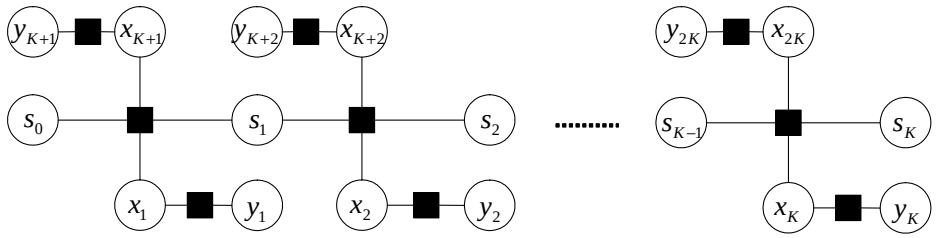

Fig. 2.11 Factor graph representing a convolutional code.

2.7.2 Sum-product Decoding

In the case of error-correcting codes, if the factor graph representing a code is tree structured, we can compute posterior probabilities of each information symbols given a received sequence by the sum-product algorithm. Then we can apply the MAP decoding to the code.

Convolutional codes can be represented by a tree structured factor graph by introducing the state variables. If the parameter r of convolutional codes, which effects the cardinality of the state variable, is small then we can apply the MAP decoding using the sum-product algorithm with low complexity. However, the minimum weight of codewords in convolutional codes with a small value of r is low. Then, the error-correcting capability of such a convolutional code is not good itself.

In recent years, a new class of codes has been widely studied; the code is decoded by using the sum-product algorithm. Turbo codes and Low-Density Parity-Check (LDPC) codes are famous for such codes.

The factor graphs representing turbo codes and LDPC codes are not tree structured. If we perform the sum-product algorithm over the non-tree structured factor graph, we can obtain *pseudo posterior probabilities* of each information symbols. Although they are not exact posterior probabilities, we can perform the *pseudo* MAP decoding that treats the pseudo posterior probabilities as the exact ones. It is shown that good decoding performance can be obtained by the pseudo MAP decoding for turbo codes and LDPC codes. We call the decoding methods using the sum-product algorithm is called the *sum-product decoding* whether or not the factor graph is tree structure.

Turbo codes and LDPC codes are defined as not a specific code but an *ensemble* of codes. Many researchers proposed code design methods for these codes to obtain good error-correcting capabilities. Moreover, it is shown that there are codes which have a weight distribution so that the good error correcting capability can be achieved by using the ML decoding in these ensemble [20] [23].

Therefore, we say they not only are good codes themselves but also have an appropriate graph representation for the sum-product decoding.

Since the factor graphs representing turbo codes and LDPC codes are not tree structured, then the computation order of the processes of the sum-product algorithm needs to be considered. Many researchers proposed the computation order to improve the decoding

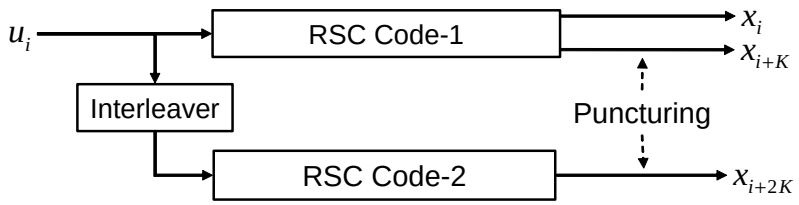

Fig. 2.12 Turbo Code.

performance. In this thesis, we treat the original ones that are proposed in [4] [20].

2.8 Turbo Codes

The original concept of turbo codes was introduced in a paper by Berrou, et. al. [4]. A turbo codes is the parallel concatenated codes of two RSC code. The encoder of turbo codes has a module called an *interleaver*, which permutes the order of symbols in an information sequence. While one RSC code encodes the original information sequence, the other RSC code encodes the permuted information sequence by the interleaver, as shown in Fig. 2.12. We denote them the RSC-1 and RSC-2, respectively.

Interleaver

The concept of design of the permutation pattern of the interleaver is so that when the weight of codeword corresponding to an information sequence is low, the weight of codeword corresponding to the permutated information sequence becomes high. Then the weight of the overall codeword is high. This design can be achieved by using the feature of RSC codes. Constructing methods of such permutation patterns of the interleaver are proposed in [23].

Puncturing

After encoding, each output is punctured; puncturing is the process of removing some codeword symbols from the codeword sequence. Puncturing is used for rate control of codes; the puncturing pattern depends on the decoding performance. In most cases, all codeword symbols corresponding information symbols of the RSC-2 are punctured.

2.8.1 Sum-product Decoding for Convolutional Codes (BCJR Decoding)

Convolutional code can be represented by a tree-structured factor graph to introduce the state variables, as shown in 2.3. Then, we can compute exact posterior probabilities of all information symbols by the sum-product decoding. The most efficient order of message processing is to perform a two-way computation; that is, the left side to the right side and the left side to the right side.

The MAP decoding using this idea was already proposed in 1974 [2]. The sum-product decoding for convolutional codes with the two-way computation is then traditionally called the *BCJR decoding*.

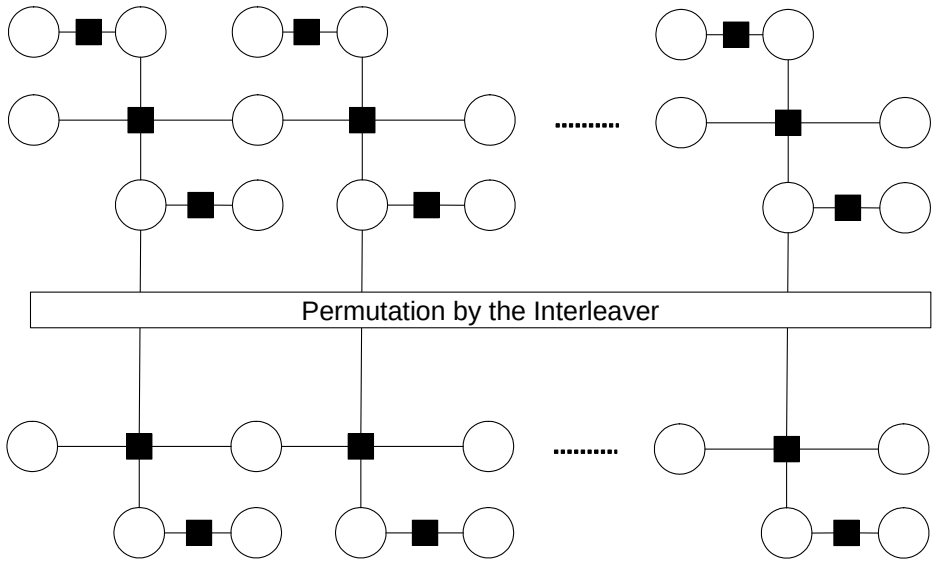

Fig. 2.13 A factor graph representing turbo code.

2.8.2 Sum-product Decoding for Turbo codes (Turbo Decoding)

In turbo code, the information sequence is independently encoded by two RSC codes; both the factor graphs representing the two codes contain the variable nodes representing same information symbols. Therefore, turbo codes can be represented by the merged factor graph of the two factor graphs by connecting the variable nodes representing information symbols via the interleaver, as shown in Fig. 2.13.

Berrou et. al. proposed the following order of processes of the sum-product decoding:

1. Perform the BCJR decoding using the received sequence corresponding to the RSC-1 over the corresponding subgraph.
2. Compute message oriented from the subgraph of the RSC-1 to the subgraph of the RSC-2.
3. Perform the BCJR decoding using the received sequence corresponding to the RSC-2 over the corresponding subgraph.
4. Compute message oriented from the subgraph of the RSC-2 to the subgraph of the RSC-1.
5. Iterate above processes until the number of iterations exceeds a predetermined value.
6. Perform the pseudo MAP decoding.

This decoding is called the *turbo decoding*.

2.9 LDPC Codes

LDPC codes are defined as linear block codes whose parity check matrix is a sparse matrix. Then the factor graph representing an LDPC code is a sparse graph accordingly. LDPC

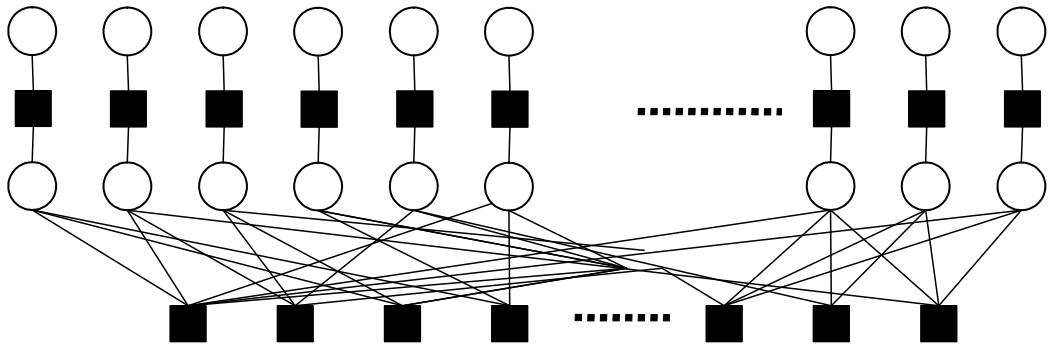

Fig. 2.14 A factor graph representing an LDPC codes.

codes were first discovered by Gallager in his Ph.D. thesis of 1960 [9] and were rediscovered independently by MacKay and Neal [20], Sipser and Spielman [31] in the mid 1990s.

The ensemble of LDPC codes are defined by degree distributions of both variable and factor nodes. If degrees of all variable nodes are same and degrees of all factor nodes are same, the LDPC code is called *regular*. Regular LDPC codes whose the degree of variable nodes is j and the degree of factor nodes is k are denoted by (j,k)-LDPC codes. Fig. 2.14 shows the factor graph representing an LDPC code.

2.9.1 Sum-product Decoding for LDPC codes

We describe the procedure of the sum-product decoding proposed in [20]. This procedure simplified the factor node process by considering binary parity constraints. [*1] Each message is computed with respect to $1 \leq n \leq N$ and $1 \leq m \leq M$.

Initialization.

For all pairs (m, n) such that $H_{mn} = 1$,

$$\begin{cases} q^0_{mn} = p(y_n|0) \\ q^1_{mn} = p(y_n|1) \end{cases} \tag{2.36}$$

Factor Node Process.

For all pairs (m, n) such that $H_{mn} = 1$,

$$\begin{cases} r^0_{mn} = ((1 + \delta r_{mn})/2) \\ r^1_{mn} = ((1 - \delta r_{mn})/2) \end{cases} \tag{2.37}$$

where

$$\delta r_{mn} = \prod_{n' \in \mathcal{N}(m) \backslash n} (q^0_{mn'} - q^1_{mn'}) \tag{2.38}$$

[*1] The sum-product algorithm defined on a logarithm domain is also proposed in [9]. In practical, it is well used because it solves over-flow problems.

Variable Node Process.

For all pairs (m, n) such that $H_{mn} = 1$,

$$\begin{cases} q^0_{mn} = \alpha(p(y_n|0) \prod_{m' \in \mathcal{M}(n)\backslash m} r^0_{m'n}) \\ q^1_{mn} = \alpha(p(y_n|1) \prod_{m' \in \mathcal{M}(n)\backslash m} r^1_{m'n}) \end{cases} \tag{2.39}$$

(α denotes the normalized operation.)

Marginalization Process.

For all pairs (m, n) such that $H_{mn} = 1$,

$$\begin{cases} q^0_{n} = \alpha(p(y_n|0) \prod_{m' \in \mathcal{M}(n)} r^0_{m'n}) \\ q^1_{n} = \alpha(p(y_n|1) \prod_{m' \in \mathcal{M}(n)} r^1_{m'n}) \end{cases} \tag{2.40}$$

Decoding Process.

For all n,

$$\hat{x}_i = \begin{cases} 0 & (q^0_n \geq q^1_n) \\ 1 & (q^0_n < q^1_n) \end{cases} \tag{2.41}$$

If $\hat{x}H^T = 0$ or the number of iterations exceeds a predetermined value , then this algorithm halts; else go to the factor node process.

2.10 Studies on Error-correcting Codes on Graphs

2.10.1 Performance Analysis

Turbo codes and LDPC codes attracted considerable attention because they can came closest to approaching the Shannon limit with long codeword length. However, a theoretical explanation for the superior performance of their decoding algorithm has not been elucidated.

In the case of the BEC, the sum-product decoding can be simplified to a similar message-passing decoding. The performance of the message-passing decoding is determined by certain combinatorial structures known as *stopping sets* [6]. We treat it in Chapter 5.

On the other hand, the performance of Turbo codes and LDPC codes using the ML decoding has been analyzed theoretically [22][23]. These studies show that turbo codes and LDPC codes satisfy good properties of codeword weight.

Asymptotic numerical analysis methods for the sum-product decoding are proposed, called the density evolution analysis [25] and the EXIT charts [5]. They indicate good code designs for these codes with long codeword length. However, the performance measure of the sum-product decoding with short and medium codeword length still have no choice but to depend on numerical experiments.

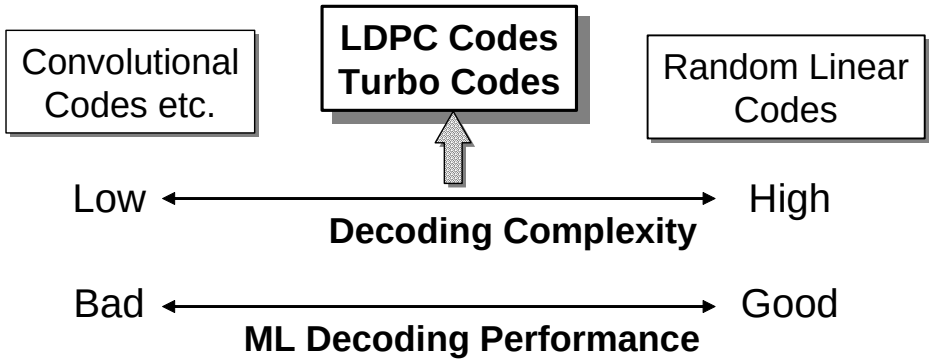

Fig. 2.15 Performance and complexity tradeoff of codes.

2.10.2 Comparison with Traditional Error-correcting Codes

Why do these codes using the sum-product decoding show excellent performance? We may provide an aspect showing relationship to traditional codes, as in Fig. 2.15.

Random linear codes [10] are defined by randomly constructed parity check matrices. The ensemble of random linear codes can achieve very low error probability using the ML decoding averagely. However, but no decoder is available with both good decoding performance and low complexity [*2].

By contrast, in the case of the convolutional codes with a small value of r, we can compute exact posterior probabilities by the sum-product algorithm. However, the minimum weight of codewords in such convolutional codes is low. Then, the error-correcting capability of such a convolutional code is not good itself.

Both turbo codes and LDPC codes are interpreted in intermediate classes of codes between random linear codes and convolutional codes. It is shown in [20] [23] [33] that the ensemble of both turbo codes and LDPC codes can achieve very low error rate using the ML decoding averagely although it does not exceed that of random linear codes. Furthermore, we can perform the pseudo MAP decoding for these codes by using the sum-product decoding with low complexity, although exact posterior probabilities cannot be computed.

We say that turbo codes and LDPC codes not only are good codes themselves but also have an appropriate graph representation for the sum-product decoding.

[*2] It is shown in [33] that the empirical performance of random codes using the sum-product decoding is bad because of the non sparseness of them factor graph.

Chapter 3

An Adaptive Encoding Scheme Using a Feedback Channel

3.1 Introduction

Consider a parallel concatenated code, which distinctly encodes the information sequence by using a number of encoders. The concatenated code can be represented by the merged factor graph of the factor graphs representing constituent codes. Therefore, the receiver can perform the sum-product decoding by using the all received sequences corresponding to each constituent code.

If the factor graph representing the concatenated code is not tree structured, then the order of message computations of the sum-product algorithm needs to be specified. One of the order of message computations is the turbo decoding, which appears that the message computations to obtain (local) exact posterior probabilities are performed in a subgraph corresponding to each constituent code, and each subgraph exchanges messages. It is shown that good decoding performance can be obtained by this order of message computations for turbo codes.

In this chapter, we consider a repeat request scheme using a feedback channel proposed by Shea in [30]. We extend this scheme by focusing on the above characteristics of parallel concatenated codes. In addition, we propose an adaptive encoding method based on the characteristics of both turbo codes and turbo decoding. We show that the complexity introduced by the adaptive feature is low and demonstrate the performance of our proposed method by carrying out numerical experiments.

3.2 Repeat Request Schemes with a Feedback Channel

We consider an error-correcting procedure using a feedback channel as shown in Fig. 3.1. The feedback channel is the channel that the transmitter can receive feedback from the receiver. For simplicity, we assume that the feedback channel has no noise and a high capacity. In this chapter, we consider schemes by which the transmitter transmits information about one information sequence several times until a termination condition is satisfied. The

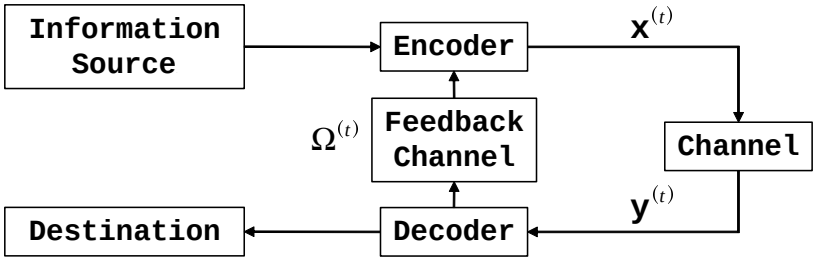

Fig. 3.1 Error-correcting procedure using a feedback.

feedback is used for the determination of the termination condition and/or the transmitting information; the later means that the encoding method is determined according to the feedback.

In this chapter, the word *time point* means the time point according to the a sequence transmission at the transmitter; that is, the time point is incremented by one whenever the transmitter transmits a sequence. We denote sequences and symbols for the time point t by the super script (t).

The automatic repeat request (ARQ) is a well known scheme using the feedback channel. In an ARQ system, a code with good error-detecting capability is used. If the receiver does not detect errors from a received sequence, a positive acknowledgment (ACK) signal is transmitted to the transmitter. If the receiver detects errors, a negative acknowledgment (NAK) signal is transmitted to the transmitter. Then the transmitter resends the same codeword. Combinations of the ARQ scheme and error-correcting codes are called hybrid-ARQ schemes. Variety of ARQ schemes and hybrid-ARQ schemes have been proposed [17].

In traditional ARQ schemes, repeat requests are performed according to block-wise decisions. Shea proposed a symbol-wise repeat requests scheme using the sum-product decoding in [30], called the *reliability-based hybrid ARQ scheme*. Shea defined a *reliability* of each information symbol after the sum-product decoding. Reliability of the information symbol u_i at a time point t is defined by

$$\lambda_i^{(t)} = \log \frac{\mathrm{app}(0)^{(t)}}{\mathrm{app}(1)^{(t)}}, \tag{3.1}$$

where app(0) and app(1) are exact or pseudo posterior probabilities computed by the sum-product algorithm.

Unreliable symbol positions are chosen from the symbol positions with low values of reliability. Two types of choices can be considered: a choice of all symbol positions with the smaller value of reliability than a threshold and a choice of predetermined fixed number of symbol positions with the smallest values of reliability. Let $\Omega(t)$ be the set of unreliable symbol positions at time point t.

In Shea's scheme, after the receiver sends the unreliable symbol positions to the transmitter, the transmitter transmits no encoded information symbols corresponding to the unreliable symbol positions. This method is a simple scheme and performs with low complexity.

3.3 Adaptive Encoding Scheme Using a Feedback Channel

We extend Shea's scheme by focusing on the characteristics of parallel concatenated codes.

First we explain it with a simple example. Consider two codes; one of code needs to adaptively and dynamically encode based on auxiliary information. We call such the code an additional code. Define a parallel concatenated code consisting on these codes. The factor graph of the concatenated code can be represented by merging the factor graphs representing the factor graphs representing two codes as shown in Fig 3.2. Because they share common variables nodes representing information symbols, and then two factor graphs can be merged connecting the common variables nodes. We can decode the concatenated code by the sum-product decoder using the merged factor graph.

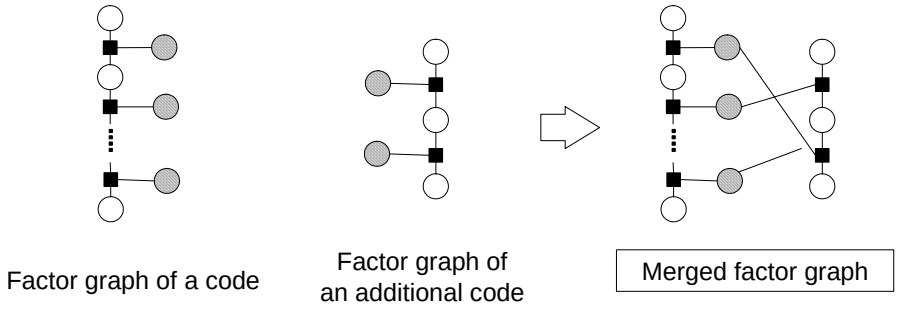

Fig. 3.2 Merging of Factor Graphs.

We extended Shea's scheme by focusing these characteristics as following:

1. At the first time point, the transmitter encodes the information sequence by a code and transmits the codeword.
2. After the receiver provides the unreliable symbol positions to the transmitter, the transmitter adaptively encodes the information sequence based on the unreliable symbol positions and transmits the codeword. This procedure can be repeated until a termination condition is satisfied.
3. The decoder estimates the information sequence using the sum-product decoding over the merged factor graph.

Additional codes needs to be adaptively and dynamically encode based on the unreliable symbol positions. We show conditions that codes using this extended scheme should be satisfied:

1. The additional codes provide higher error-correcting capability for the unreliable symbol positions than the other positions.
2. The complexity of encoding and decoding methods due to the dynamic behavior is as small as possible.
3. The factor graphs representing concatenated codes at each time point are appropriate to the sum-product decoding.

3.4 Adaptive Encoding Method Based on Turbo Codes

We propose an adaptive encoding method based on the characteristics of both turbo codes and turbo decoding. For simplified description and analysis, we show the procedure that the transmitter transmits only two codewords. It is easily extended to the case when the transmitter transmits more codeword.

1. At the first time point, the transmitter encodes the information sequence by an RSC code and transmits the codeword.
2. The receiver computes reliabilities using the BCJR decoding and provides unreliable symbol positions to the transmitter.
3. At the second time point, the transmitter decides both the permutation pattern of the interleaver and the puncturing pattern based on the unreliable symbol positions. The transmitter encodes the information sequence by an RSC codes using both this interleaver and this puncturing pattern, and transmits the codeword.
4. The decoder estimates the information sequence using the turbo decoding over the merged factor graph.

This method is shown in Fig. 3.3.

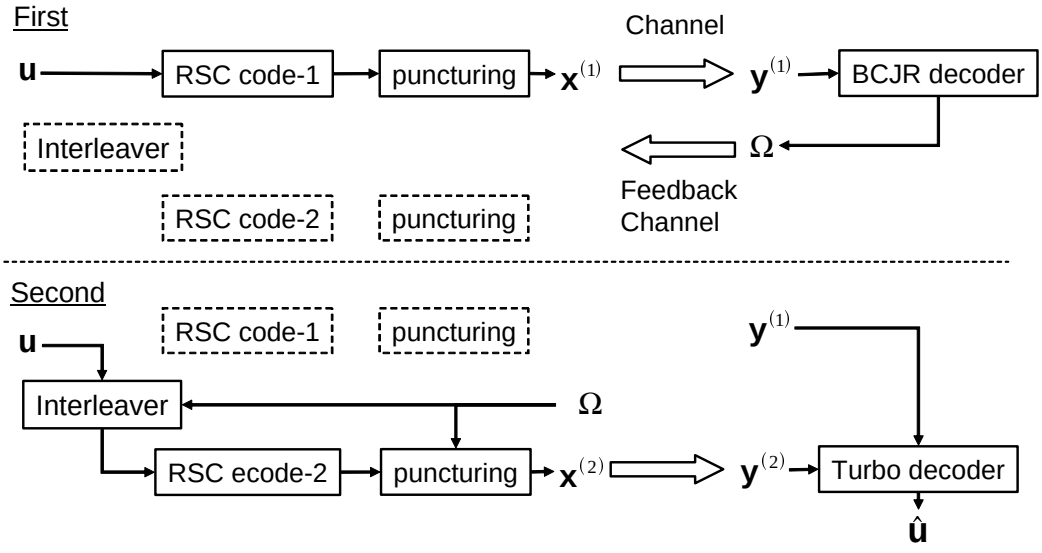

Fig. 3.3 Adaptive encoding method based on turbo codes.

We show this method satisfies the above stated conditions. The first condition can be satisfied by appropriate designs of the permutation pattern of the interleaver and the puncturing pattern. We will propose these designs in the following subsections.

The dynamic behaviors according to unreliable symbol positions in our proposed method are only design of the permutation pattern of the interleaver and the puncturing pattern. These patterns can be dynamically modified with low complexity. The main constituents of

turbo codes, which are both the RSC encoders and the BCJR decoders, are not modified. Then, the second condition is satisfied.

Since the decoding at the first time point is the BCJR decoding, the exact posterior probabilities can be computed. This is good feature to compute the reliabilities. Moreover, the concatenated code of the two codes is similar as a turbo code, and the decoding at the second time point is the turbo decoding. It is shown that good decoding performance can be obtained by the turbo decoding for turbo codes. Then the third condition is satisfied.

3.4.1 Adaptive Design of the Permutation Pattern of the Interleaver

This subsection describes an adaptive design of the permutation pattern of the interleaver based on the unreliable symbol positions.

Error-correcting codes that protect some positions in the information sequence against a larger number of errors than other ones are called unequal error protection codes [11]. We consider a special class of unequal error protection codes to satisfy that an RSC code provides higher error-correcting capability for the unreliable symbol positions than the other positions.

Let $\pi(i)$ be the permutation function so that the i-th information symbol is permutated to the $\pi(i)$-th symbol of the output sequence form the interleaver. Define a subset of $\{0,1\}^K$ based on the unreliable symbol positions Ω :

$$\omega(\Omega) := \left\{ \mathbf{u}' \in \{0,1\}^K \mid u'_{\pi(i)} = 0, \quad \forall i \notin \Omega \right\}. \tag{3.2}$$

This subset has the following mean: consider a binary sequence $\mathbf{u}$ which is a member in $\omega(\Omega)$. If we flip a number of symbol positions of the sequence $\mathbf{u}$, which positions are indicated in Ω, the flipped sequence is also the member of $\omega(\Omega)$.

If we find a code in which all codewords corresponding to $\mathbf{u} \in \omega(\Omega)$ have high weight, the code provides higher error-correcting capability for the unreliable symbol positions than the other positions.

To find the best permutation pattern of the interleaver for an RSC code, we need to evaluate the weight distribution of codewords corresponding all members in $\omega(\Omega)$. It is very hard because the evaluation is needed for various patterns of the symbol positions Ω.

Alternatively, we propose the following simple permutation patterns considering only the minimum weight. Let $\pi'(i, B)$ be a permutation function satisfying that the all interval between the adjacent output positions corresponding to the positions in the Ω are B. Fig. 3.4 shows an example of this permutation function.

Consider the RSC code with the generator matrix $[1, (1+D^3)/(1+D+D^2+D^3)]$. All information sequences with weight 2 such that the interval between two symbols "1" is 3 is encoded to weight 4 codewords. For example, $\mathbf{u}$ = ...00010001000... is encoded to $\mathbf{x}$ = ...00011011000.... Then, $\pi'(i, 3)$ is not appropriate for this RSC code.

On the other hand, all information sequence with weight 2 such that the interval between two symbols "1" is 2 is encoded to infinite (if not terminating) weight codewords. For example, $\mathbf{u}$ = ...0001001000... is encoded to $\mathbf{x}$ = ...0001100111111.... We consider an information sequence with low weight such that the interval between two symbols "1" is 2; the weight

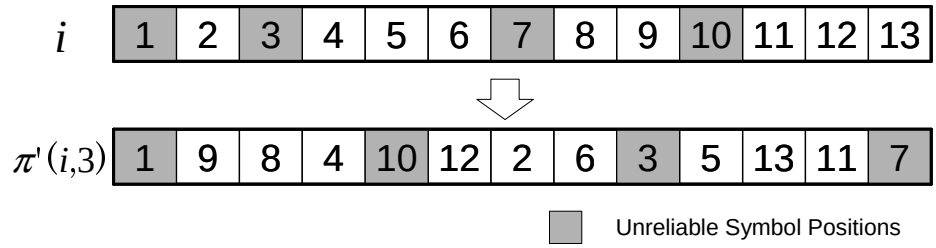

Fig. 3.4 An example of the permutation function $\pi'(i, 3)$.

of such the information sequence is 4. For example, $\mathbf{u}$ = ...00100100100100... is encoded to the weight-6 codeword $\mathbf{x}$ = ...00110011001100....

Then, the combination of this RSC code and the permutation function $\pi'(i, 2)$ provides higher error-correcting capability for the unreliable symbol positions than the other positions. They are appropriate for our proposed method.

Furthermore, the combination of the RSC code with the generator matrix $[1, (1+D^4)/(1+D+D^2+D^3+D^4)]$ and the permutation function $\pi'(i, 3)$ is appropriate for our proposed method, for similar reasons.

3.4.2 Adaptive Design of the Puncturing Pattern

This subsection describes an adaptive design of the puncturing pattern based on the unreliable symbol positions.

A factor graph can present the strength of relation between pairs of codeword (or information) symbols as the distance between the corresponding variable nodes. Focusing on the characteristic, we propose the following dynamic puncturing pattern design for the RSC code at the second time point : only the following codeword symbols are transmitted, such that they correspond to the input of the unreliable symbol positions of the pre-interleaved information sequence.

We show the reasons why this puncturing pattern is a good design:

1. Transmitted codeword symbols are allocated near unreliable information symbols. New information of the unreliable information symbols is provided themselves. Then the performance of the BCJR decoding for the RSC code at the first time point can be improved.
2. Punctured codeword symbols are allocated near reliable information symbols. Assuming that reliable symbols have been correctly estimated at the receiver, it provides that the performance of the BCJR decoding for the RSC code at the second time point can be improved.

This is illustrated in Fig. 3.5. The figure omits interleaving and the variable nodes corresponding received symbols. Note that the assumption in the second reason is not always true. Because there are cases that the reliability may take a high value even if the estimation for the information symbol is not correctly estimated.

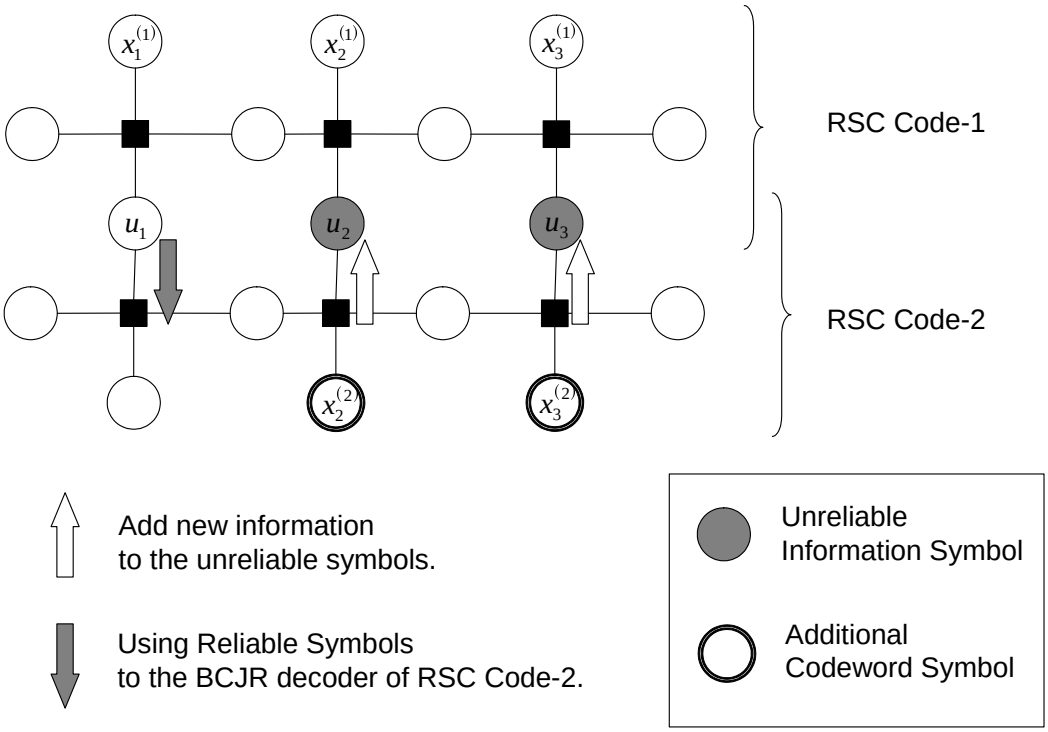

Fig. 3.5 Adaptive design of the puncturing pattern.

3.5 Numerical Experiments

We evaluate the decoding performance of our proposed method by numerical experiments. We compare the decoding performance of our proposed method to the decoding performance of the similar turbo code without using a feedback channel. This provides the validity of our proposed adaptive designs of the interleaver and puncturing pattern.

The following codes are used:

- (Code A) The turbo code including the same two RSC codes with the generator matrix $[1, (1 + D^3)/(1 + D + D^2 + D^3)]$, the interleaver length of 300, and the puncturing interval is $P = 3$.
- (Code B) The turbo code including the same two RSC codes with the generator matrix $[1, (1 + D^4)/(1 + D + D^2 + D^3 + D^4)]$, and the interleaver length of 400, and the puncturing interval is $P = 4$.

Termination process of the convolutional code is performed only for the RSC code at the first time point. The puncturing rule for the RSC encoder at the first time point is to puncturing all i-th codeword symbols, where $i \mod P \neq 0$. The number of unreliable symbol positions provided by the BCJR decoder is 100. We evaluate the following two variations of the interleaver. One uses a random interleaver (denoted by Procedure 1); the other uses the $\pi'(i, 2)$ for Code A and $\pi'(i, 3)$ for Code B (denoted by Procedure 2). The

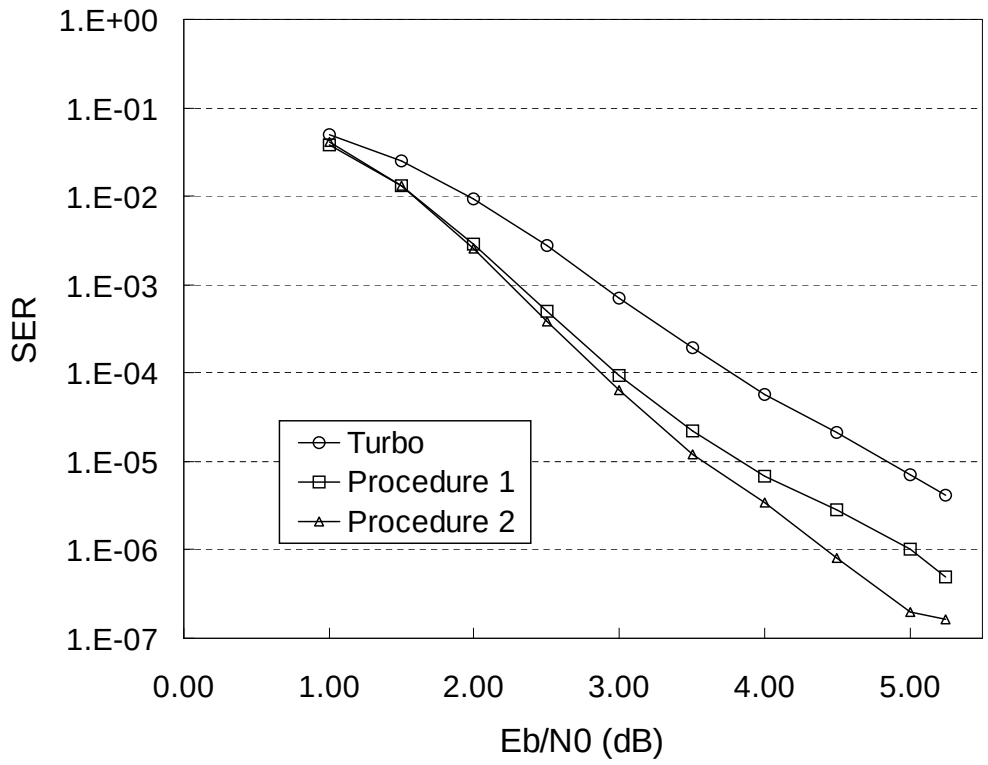

Fig. 3.6 Code A result.

BIAWGN channel is considered. Figures 3.6 and 3.7 show the symbol error rates (SER) for Code A and Code B, respectively.

3.6 Discussions

Decoding Performance

The results show that the decoding performance of our proposed method is much better than the turbo code that has similar constitution codes. It is shown that our proposed dynamic decisions of the interleaver and puncturing pattern are appropriate for our proposed scheme.

Comparing the performances between the Procedure 1 and the Procedure 2, our proposed dynamic decision of the interleaver is very effective in the low noise region. It is caused by avoiding low weight codewords corresponding to the information sequences $\mathbf{u} \in \omega(\Omega)$.

Complexity

The dynamic behaviors according to unreliable symbol positions in our proposed method are only design of the permutation pattern of the interleaver and the puncturing pattern. These patterns can be dynamically modified with low complexity. The main constituents of turbo codes, which are both the RSC encoders and the BCJR decoders, are not modified. Then, the complexity of encoding and decoding methods due to the dynamic behavior is

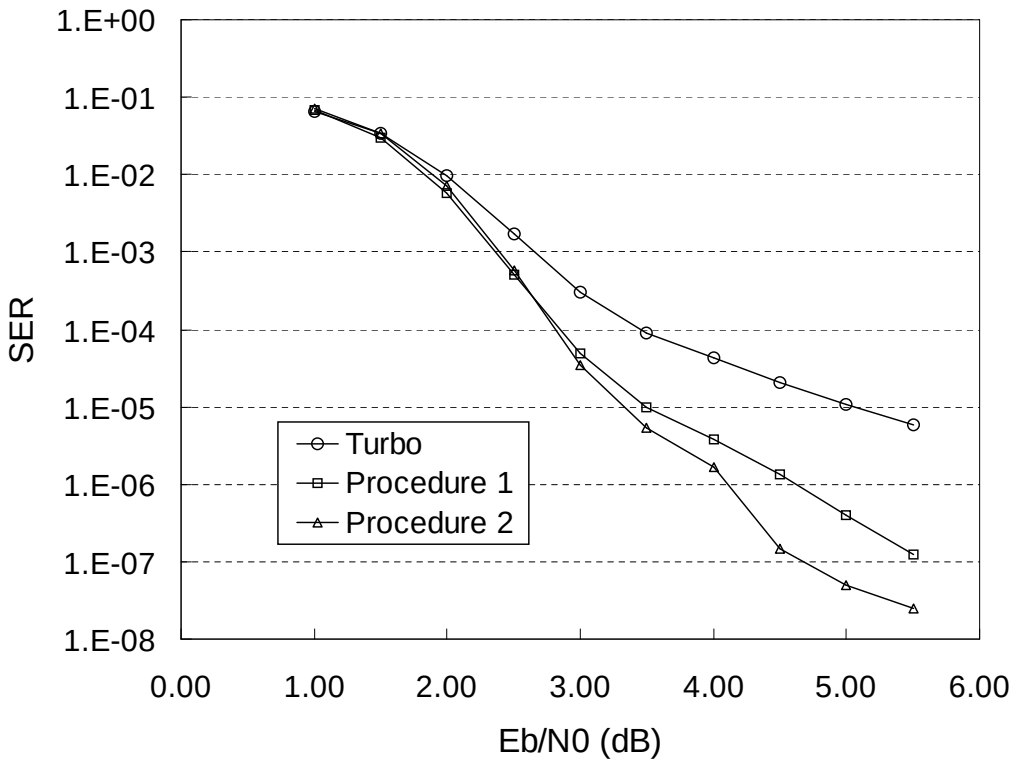

Fig. 3.7 Code B result.

low.

3.7 Concluding Remarks

We extended the Shea's scheme by focusing on the characteristics of parallel concatenated codes and proposed an adaptive encoding method based on the characteristics of both turbo codes and turbo decoding. We showed that the complexity introduced by the adaptive feature is low and demonstrated the performance of our proposed method by carrying out numerical experiments.

Our proposed method utilizes the characteristics of graph representation of codes; that is, the characteristic of merged factor graph representing parallel concatenated codes is used as the basis of our proposed repeat request schemes, and used for the adaptive design of the interleaver. Moreover, the characteristic such that a factor graph can present the strength of relation between pairs of symbols as the distance between the corresponding variable nodes, is used for the adaptive design of the puncturing pattern.

Chapter 4

A Transformation of a Factor Graph for LDPC Codes with Small Cycles

4.1 Introduction

A code can be represented by several factor graph representations; that is, a code and a factor graph is not one-to-one relation. We can transform a factor graph representation to another one by some method. This chapter and the next chapter focus on this characteristic.

In this chapter, we use the *clustering method* proposed by J. Pearl in the study of artificial intelligence [24]. The clustering method was proposed to transform a non-tree structured graph representing a probabilistic model, called the Bayesian network, to a tree structured one. First, we propose a scheme of the sum-product decoding using the factor graph transformed by the clustering method.

Besides, many researchers have been reported that the empirical performance of the sum-product decoding depends on the cycles in the factor graph. For LDPC codes, it is shown cycles with small length, especially of the length 4, have a negative effect of the decoding performance [20].

By considering transformations of a factor graph, we approach the problem from a point of view of designing the decoder without changing the code itself. We propose a method to transform only subgraphs corresponding small cycles by the clustering method. We show by the numerical experiments that the degradation of the decoding performance according to small cycles can be mitigated by our proposed method.

4.2 Transformations of a Factor Graph

A code can be represented by several factor graph representations. A notable example is factor graph representations convolutional codes. The parity check matrix of convolutional codes can be obtained from the generator matrix, and then we can represent the factor graph by the form of block linear codes. Meanwhile, by introducing variables denoting the states, we can represent the factor graph by a tree-structured form. We can compute exact posterior marginal distribution of each symbol by the sum-product decoding using the

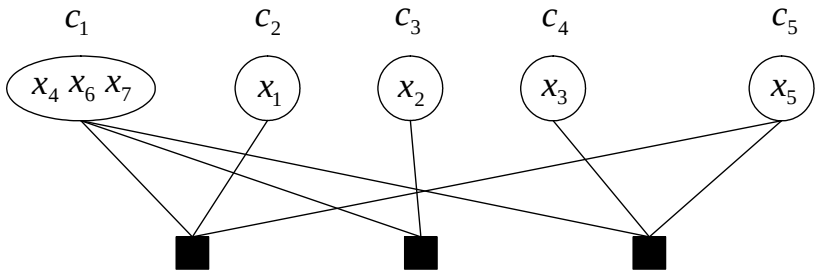

Fig. 4.1 An example of a clustered factor graph.

later factor graph. However, we cannot do it using the former factor graph. It shows that the factor graph representation of the code depends on the procedure of the sum-product decoding and also its decoding performance.

By some methods, we can transform a factor graph representation to another one. Some transformation methods were proposed in [16].

4.3 Clustering Method

The clustering method [24] was proposed in the study of artificial intelligence. The clustering method is to transform a non-tree structured graph representing a probability model, called the Bayesian network, to a tree structured one. We apply the method to factor graphs.

By the clustering method, a graph can be transformed by introducing a random variable which is defined by the direct product of several random variables; it yields that several variable nodes are combined into a single variable node on the graph.

For example, consider the factor graph in Fig. 2.10. If variables x_4, x_6, and x_7 are dealt with a single variable, the factor graph can be represented by Fig. 4.1. The domain of the combined variable is defined by the combination of the domains of the variables x_4, x_6 and x_7.

We call a factor graph to which the clustering method applies a *clustered factor graph* and call a new node provided by combining a *cluster node*. Cardinality of a variable corresponding to a cluster node increases exponentially as the number of combined nodes increases. Then the complexity of message computation of the sum-product algorithm also increases exponentially as the number of combined nodes increases.

We show some notations for a clustered factor graph. Let $k(1 \leq k \leq N_c)$ be an index of each cluster node, where let N_c denotes the number of cluster nodes in a clustered factor graph. The k-th cluster node is represented by c_k. Let the vector C_k denotes the indices of combined variables of the original factor graph into the cluster node c_k. We have $C_i \cap C_j = \phi$ if $i \neq j$. The t-th element of C_k is denoted by C_{kt}. When in the case that a symbol of the original factor graph is not combined, we treat it as a cluster node with $|C_k| = 1$.

We should define new variables corresponding to cluster nodes, but however we identify the clustering node c_k as the variable c_k for simplicity. Let $A_{c_k} \in \{0,1\}^{|C_k|}$ be the domain of the variable c_k. Let $M'(k)$ be the set of all factor nodes connecting to the cluster node

c_k, and we have

$$M'(k) = \bigcup_{t=1}^{|C_k|} M(C_{kt}). \tag{4.1}$$

Let $N'(m)$ be the set of all variable nodes connecting to the m-th factor node of a clustered factor graph. Since not all variable nodes in the original factor graph connect the factor nodes connecting to the corresponding cluster node, we define the following set to represent such a relation:

$$D_{km} := \{n | n \in C_k \cap N(m)\}. \tag{4.2}$$

For example, for the cluster factor graph as shown in Fig. 4.1, we have $N'(1) = \{1, 2, 4\}$, $N'(2) = \{1, 3\}$, $N'(3) = \{1, 4, 5\}$, $C_1 = \{4, 6, 7\}$, $C_2 = \{1\}$, $C_3 = \{2\}$, $C_4 = \{5\}$, and $C_5 = \{3\}$. Furthermore, $D_{11} = \{4, 7\}$ is given from $C_1 = \{4, 6, 7\}$, and $N(1) = \{1, 4, 7\}$.

4.4 Clustering Method and Factor Graphs Representing LDPC Codes

We want to get a tree-structured factor graph to compute exact posteriori probabilities. However, in the case of LDPC codes, to obtain a tree-structured factor graph, almost all nodes in the factor graph need to be combined. This results in exponential increase in the complexity of the sum-product decoding. It is not practical. Therefore, we propose a method in which only the subgraphs of the factor graph are transformed by the clustering method.

Our proposed method implies that the accuracy of the computation of approximate posterior probabilities increases by increasing the complexity of the sum-product algorithm of the subgraph. A concept of our proposed method is illustrated as in Fig. 4.2. The leftmost of the figure corresponds to the sum-product decoding using the original factor graph of an LDPC code. On the other hand, the rightmost corresponds to the sum-product decoding using the tree-structured factor graph transformed by the clustering method. We can and need to consider the balance between the complexity and accuracy of the sum-product decoding by using our proposed method.

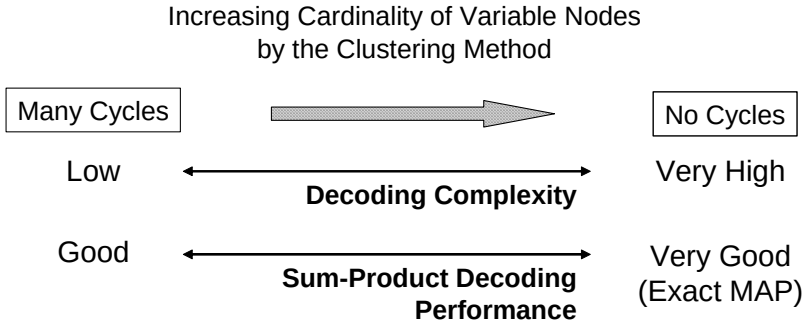

Fig. 4.2 Relationship of cycles and complexity by the clustering method.

First, we present message computation rules of the sum-product decoding using a clustered factor graph. By the message computation rules, we can decrease the complexity of the sum-product decoding using a clustered factor graph.

4.4.1 Message Computation Rule for Clustered Factor Graphs

Messages from a cluster node c_k to a factor node h_m are defined for each element in $A := \{0,1\}^{D_{km}}$. If $|D_{km}| \neq |C_k|$, it does not coincide with elements of the another message to the cluster node c_k. Then, when the message for an element $a \in A$ is computed, we need a marginal operation of another messages over all elements corresponding to the element a.

For efficient computation, we compute additional values (represented Q in the algorithm) before computing messages from a cluster node unlike the original sum-product algorithm. Although the number of elements of a message of the sum-product algorithm between a cluster node and the neighbor factor node equals the number of elements of the domain for the cluster node, we compute only two elements of a message the *even-message* and the *odd-message*. This efficient computation is derived the fact that that the factor node menas the parity-check constraint.

For example, the left factor node in the Fig. 4.1 represents

$$X_1 + X_4 + X_5 + X_7 = 0 \mod 2. \tag{4.3}$$

For messages between this factor node and c_1, all elements corresponding to ($X_4 + X_7 = 0 \mod 2$) take an equal value because of the characterization of the parity-check constraint. Then, we need only one computation, which corresponds to the even-message. We can apply it to the odd-message.

4.4.2 Sum-product Decoding Using Clustered Factor Graphs

Let q^a_{km} and r^a_{mk} be messages between the factor node h_m and the cluster node c_k, where a is a type of messages, *even* or *odd* unlike the original sum-product decoding. For any set of symbol positions B, let $\lambda_e(C_k, B)$ be the subset of $\{0,1\}^{|C_k|}$, where the symbols corresponding to the symbol positions of B have even weight, and let $\lambda_o(C_k, B)$ be also the subset of $\{0,1\}^{|C_k|}$, where the symbols corresponding to the symbol positions of B have odd weight.

For example, for the factor graph shown in Fig. 4.1, we have

$$\lambda_e(C_1, D_{11}) = \{000, 010, 101, 111\}, \tag{4.4}$$

$$\lambda_o(C_1, D_{11}) = \{001, 100, 011, 110\}. \tag{4.5}$$

Let t be the binary sequence with any length. Let t_l denotes the l-th element of the sequence t. Each message is computed with respect to $1 \leq k \leq N_c$ and $1 \leq m \leq M$.

Initialization.

For all pairs (m, k) such that $k \in N'(m)$,

$$\begin{cases} q_{km}^{even} & = \sum_{t \in \lambda_e(C_k, C_k)} \prod_{l=1}^{|C_k|} p(y_{C_{kl}} | t_l), \\ q_{km}^{odd} & = \sum_{t \in \lambda_o(C_k, C_k)} \prod_{l=1}^{|C_k|} p(y_{C_{kl}} | t_l). \end{cases} \tag{4.6}$$

Factor Node Process.

For all pairs (m, k) such that $k \in N'(m)$,

$$\begin{cases} r_{mk}^{even} & = ((1 + \delta r_{mk})/2), \\ r_{mk}^{odd} & = ((1 - \delta r_{mk})/2), \end{cases} \tag{4.7}$$

where

$$\delta r_{mk} = \prod_{k' \in N'(m) \backslash k} (q_{k'm}^{even} - q_{k'm}^{odd}). \tag{4.8}$$

If $|N'(m)| = 1$, $r_{mk}^{even} = 1$ and $r_{mk}^{odd} = 0$.

Variable Node Process.

For all pairs (m, k) such that $k \in N'(m)$ and for all $t \in \{0, 1\}^{|C_k|}$,

$$Q_k^t = \prod_{l=1}^{|C_k|} p(y_{C_{kl}} | t_l) \prod_{m \in M'(n)} r_{mk}^{a_{km}(t)}, \tag{4.9}$$

where

$$a_{km}(t) = \begin{cases} \text{“}even\text{”} & t \in \lambda_e(C_k, D_{km}), \\ \text{“}odd\text{”} & other. \end{cases} \tag{4.10}$$

Compute for each $m \in M'(k)$,

$$\begin{cases} q_{km}^{even} & = \alpha(\sum_{t \in \lambda_e(C_k, D_{km})} Q_k^t)/r_{mk}^{even}, \\ q_{km}^{odd} & = \alpha(\sum_{t \in \lambda_o(C_k, D_{km})} Q_k^t)/r_{mk}^{odd}. \end{cases} \tag{4.11}$$

Marginalization Process.

For all n such that $n \in C_k$,

$$\begin{cases} q_n^0 = \alpha \sum_{t \in \lambda_e(C_k, \{n\})} Q_k^t, \\ q_n^1 = \alpha \sum_{t \in \lambda_o(C_k, \{n\})} Q_k^t. \end{cases} \tag{4.12}$$

4.4.3 Complexity for Message Computations

We compare the number of the arithmetic operation in an iteration of our proposed decoding to the original sum-product decoding. It is shown in the following tables.

Table. 4.1 The number of the arithmetic operation in an iteration of original sum-product decoding.

Factor Node Process	$\sum_{m=1}^{M} \|N(m)\|(2\|N(m)\|+1)$
Variable Node Process	$\sum_{m=1}^{M}(\sum_{n\in N(m)}(2\|M(n)\|+1))$
Marginalization Process	$(5N)$ $(*)$

(*) This formula is derived if the sum-product algorithm uses the pre-normalization values on the variable node process.

Table. 4.2 The number of the arithmetic operation in an iteration of our proposed decoding.

Factor Node Process	$\sum_{m=1}^{M} \|N'(m)\|(2\|N'(m)\|+1)$
Variable Node Process	$\sum_{m=1}^{M}(\sum_{k\in N'(m)}(2^{\|C_k\|}(\|M'(k)\|+\|C_k\|-1)+3))$
Marginalization Process	$\sum_{k=1}^{N_c}(\|C_k\|(2^{\|C_k\|}+1))$

4.5 LDPC codes with Small Cycles

Many studies have reported that the empirical performance of LDPC codes using the sum-product decoding depends on the length of the cycles in the factor graph. It was demonstrated that cycles having small lengths, particularly those with a length of 4, have a negative effect on the decoding performance [20]. To overcome this negative effect, the use of LDPC codes, which do not have small cycles has been proposed [13] [15]. It should be noted that this approach involves the designing of codes and thus would introduce restrictions on the design of the codes. On the other hand, we can solve this problem by transformations of a factor graph without changing the code itself.

4.5.1 Clustering Method to Small Cycles

Consider the subgraph of a factor graph, which is corresponding to a cycle. If we can transform the subgraph a tree structured subgraph as shown in Fig 4.3, the message computation only in the subgraph corresponds to compute the exact posterior probabilities. As stated above, this transform implies that the accuracy of the computation of approximate posterior probabilities increases by increasing the complexity of the sum-product algorithm of the subgraph for the factor graph.

By considering this concept, we apply the clustering method to the following combinations of variable nodes, which correspond to a small cycle.

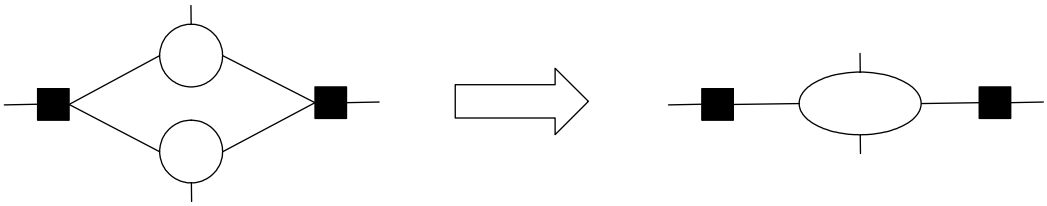

Fig. 4.3 Transform a subgraph to a tree structure.

For Small Cycles with the Length 4

Combine two variable nodes corresponding to a small cycle with the length 4 as shown in Fig. 4.4.

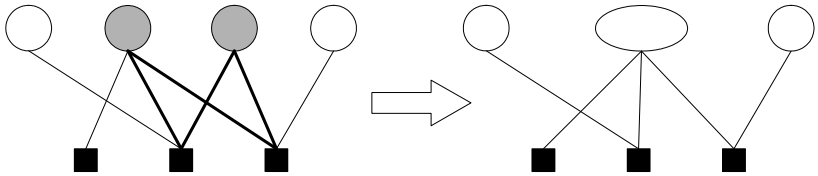

Fig. 4.4 For small cycles with the length 4.

For Small Cycles with the Length 6

Combine three variable nodes corresponding to a small cycle with the length 6 as shown in Fig. 4.5.

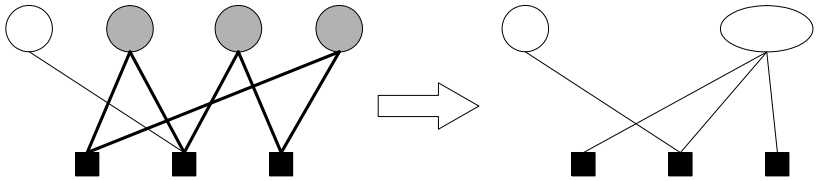

Fig. 4.5 For small cycles with the length 6.

4.5.2 Selection of Small Cycles with the Length 6

Commonly used LDPC codes have a number of small cycles with the length 6, and there are variable nodes which are shared by some cycles. Then, we need to decide a selection rule for small cycles with the length 6 to which the clustering method is applied.

We propose a rule such that the made-up clustered factor graph has no cycles with the length 4 and show the procedure as following. The procedure outputs the combination of combining variable nodes for each cluster node C_k.

Let $M(\{x_i\}) = \cap_{n\in\{x_i\}} M(n)$.

L is family of all sets of indeces of three variable nodes which contained in a cycle with the length 6.

$\forall_i S_i := 0\ (1 \le i \le N); \quad \forall_j T_j := 0\ (1 \le j \le |L|);$
$k := 1;$
for $i = 1$ to $|L|$ do {
  for $j = i + 1$ to $|L|$ do {
    if($|L_i \cap L_j| \ge 2$) then $T_i := 1;\ T_j := 1;$
  }
}
for $i = 1$ to $|L|$ do {
  if $T_i = 0$ and $S_{L_{i1}} = 0$ and $S_{L_{i2}} = 0$ and $S_{L_{i3}} = 0$ then {
    $C_k = L_i; \quad k = k + 1;$
    $S_{L_{i1}} := 1; \quad S_{L_{i2}} := 1; \quad S_{L_{i3}} := 1;$
    for $j = 1$ to $|L|$ do {
      if $|M(L_i) \cap M(L_j)| \ge 2$ then $T_j := 1$
  }
}
for $i = 1$ to N do {
    if $S_i = 0$ then { $C_k = \{x_i\}; \quad k = k + 1;$}
}

4.6 Numerical Experiments

We evaluate the performance of the proposed sum-product decoding by the following computer simulations. The BIAWGN channel is considered.

For Small Cycles with the Length 4

Consider the (3,6)-regular LDPC codes ensemble with codeword length $N = 200$ satisfying that any two columns and any two rows of a parity-check matrix don't have greater than *two* symbol "1" symbols in common. We randomly select five codes from this ensemble, and for each code we get clustered factor graphs of which the all pair of nodes for small cycles with the length 4 is combined. For both the sum-product decoding with the original factor graph and the proposed sum-product decoding with the clustered factor graph, we evaluate the decoding performance. 6×10^6 words are transmitted for each E_b/N_0. The maximum number of iterations of the sum-product algorithm is 400. The block error rate (BER) is shown in Fig. 4.6.

[Hu05] denotes that the sum-product decoding result for the (3,6)-near regular LDPC code [*1], and codeword length $N = 200$ constructed by the method in [13], whose factor graph does not contain small cycles with the length 4. In this experiments, no decoding error is occurred for $E_b/N_0 > 4.5$.

[*1] LDPC codes constructed Hu's method has no guarantee that the LDPC code is regular, that is, all variable node degree are 3, but not all factor node degree is 6.

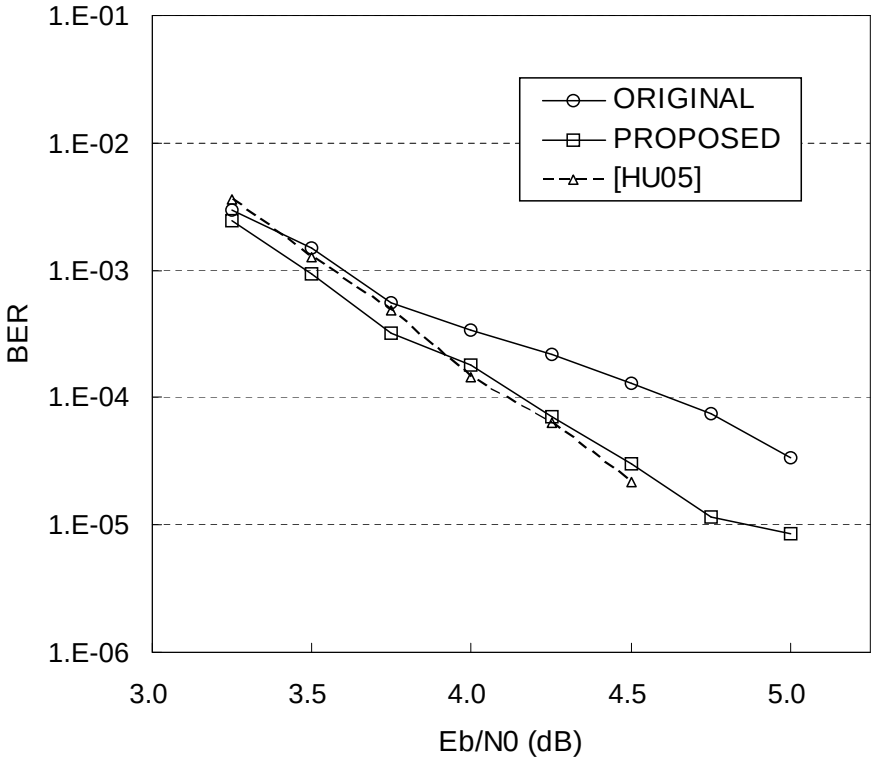

Fig. 4.6 Result for small cycles with the length 4.

For Small Cycles with the Length 6

We evaluate the (3,6)-regular LDPC code in [21] with codeword length $N = 1008$, which has no small cycles with the length 4. We get the clustered factor graph of which triplet nodes selected by the method in subsection 4.5.2 are combined. 2×10^4 words are transmitted for each E_b/N_0. The maximum number of iterations of the sum-product decoding is 100. The block error rate (BER) is shown in Fig. 4.7.

To evaluate the complexity of the proposed sum-product decoding, we show the number of the arithmetic operation (addition and multiplication) in an iteration of each decoding algorithm for this LDPC code in the Table 4.3.

Table. 4.3 The number of arithmetic operations in one iteration.

	Original	Proposed
Factor Node Process	39312	38166
Variable Node Process	21168	58674
Marginalization Process	5040	4788
total	65520	101628

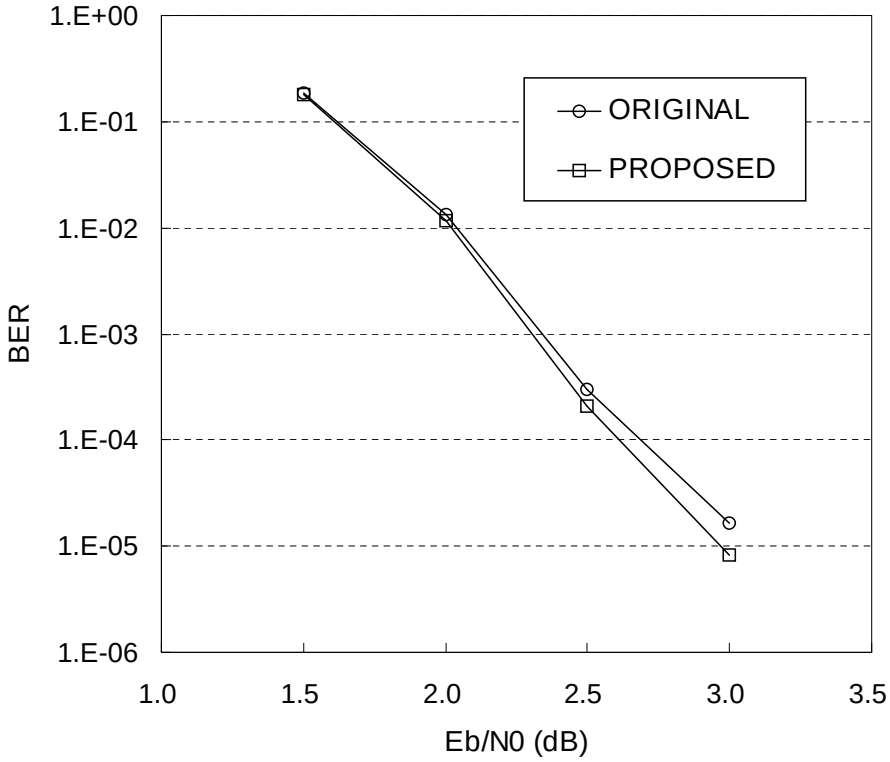

Fig. 4.7 Result for small cycles with the length 6.

4.7 Discussion

The first result shows that the proposed sum-product decoding can improve the error rate for the original sum-product decoding when the factor graph contains many cycles with the length 4. Our proposed decoding is effective when LDPC codes with the length 4 cycles are used.

On the other hand, if we can find a good LDPC code cycles with the length 4, our proposed sum-product decoding will possess great significance. It is predicted from the first result to exist such a code. Figure 4.6 shows that the error rate of the code constructed by the Hu's method is much lower than that of the codes with cycles with the length 4 in the low noise region. In the high noise region, the error rate of the code constructed by the Hu's method is a little higher than that of the codes with cycles with the length 4. We can predict that although LDPC codes with cycles with the length 4 have only low minimum weight, these may have a good weight distribution that is appropriate for the error-correcting in the high noise region.

Moreover, it may occur when cycles with the length 4 cannot be eliminated by the Hu's method in the case that the code has a short codeword length and high rate. In such case, our proposed sum-product decoding can improve the decoding performance.

The second result show that our proposed sum-product decoding can slightly improve

the error rate for the original sum-product decoding when cycles with the length 6 are combined. Table 4.3 shows that the number of arithmetic operations of the proposed sum-product decoding is less than the twice of the number of that of the original sum-product decoding. We can balance the trade-off between the improvements the performance and the complexity of the sum-product decoding.

4.8 Concluding Remarks

In this chapter, we dealt with the clustering method, which was proposed by Pearl in the study of artificial intelligence for a graph called the Bayesian network. We proposed a scheme of the sum-product decoding using the factor graph transformed by the clustering method.

Furthermore, we proposed a method to transform only subgraphs corresponding small cycles by the clustering method. We showed by the numerical experiments that the degradation of the decoding performance according to small cycles could be mitigated by our proposed method.

Chapter 5

A Transformation of a Factor Graph for LDPC Codes on the BEC

5.1 Introduction

The previous chapter described that a transformation of the factor graph representation changes the procedure of the sum-product decoding and its decoding performance on the BIAWGN channel.

In this chapter, we consider the BEC. The sum-product algorithm on the BEC can be simplified to a message-passing algorithm. The performance of the message-passing decoding is determined by certain combinatorial structures known as *stopping sets* [6]. Stopping sets are defined not on a code but on a factor graph, and then transformations of the factor graph representation change its decoding performance.

We propose a new transform method of factor graph representing LDPC codes and show that there are channel erasure patterns that can be corrected by the message-passing decoding over the transformed factor graph and that cannot correct by the message-passing decoding over the original factor graph. We prove the condition of such channel erasure patterns. Moreover, we show a relation between the transformation and the subgraphs corresponding to small cycles. By carrying out numerical experiments, we have verified that our method can improve the performance of the sum-product decoding with LDPC codes having small cycles on the BEC.

5.2 ML Decoding on the BEC

Decoding of linear block codes over the BEC is equivalent to solving simultaneous equations which is defined by its parity-check matrix. The receiver treats the positions of the codeword symbols which received as erasure as unknown variables, and obtain their values by solving the simultaneous equations by using the other known variables. Therefore, the ML decoding can be performed by using the Gaussian elimination. However, the complexity is $O(N^3)$.

On the other hand, the sum-product decoding on the BEC can be simplified to a message-

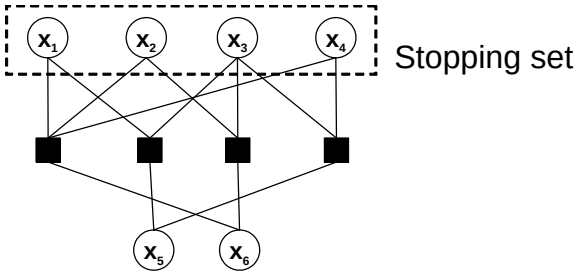

Fig. 5.1 Factor graph representing the parity-check matrix in Eq. (5.1).

passing decoding, as stated below. The complexity of the message-passing decoding is $O(N)$.

5.3 Stopping Sets

Definition 1 (Stopping Sets)**.** *A stopping set is a set of variable nodes with the property that every factor node connected to a variable node in the stopping set is connected to at least two such nodes.*

Stopping sets play an important role of the message-passing decoding of LDPC codes on the BEC. Di et. al. [6] proposed finite-length analysis of LDPC codes on the BEC using stopping sets. For the parity-check matrix in Eq. (5.1), $\{x_1, x_2, x_3, x_4\}$ is a stopping set.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}. \tag{5.1}$$

5.4 Transformations of a Factor Graph based on a Parity-check Matrix Transformation

Consider an M by N parity-check matrix for a code. Rows of a parity-check matrix indicate linear constraints of the code, and then we can add rows which do not break the linear dependency of the code into the parity-check matrix. Moreover, introducing *dummy symbols* of a codeword which are not transmitted and have new linear constraints with codeword symbols, we can also add columns which do not break the linear dependency of the code into the parity-check matrix.

When a parity-check matrix for the code has more than N columns (and more than M rows), the parity-check matrix is called a Generalized Parity-Check (GPC) matrix [34]. We denote the number of columns and rows of a GPC matrix by N' and M', respectively.

For example, the parity-check matrix in Eq. (5.1) can be transformed into the GPC matrix in Eq. (5.2). We introduce a dummy symbol corresponding to the seventh column and add new linear constraint corresponding to the fourth row. This transformation does not break

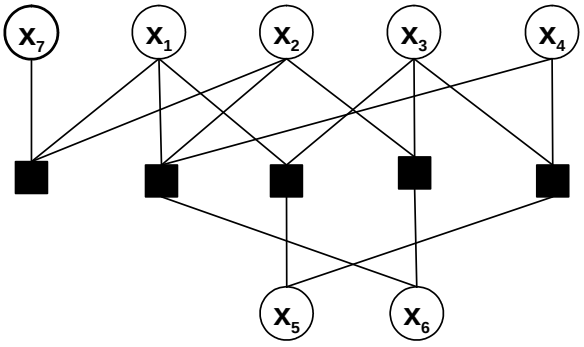

Fig. 5.2 Factor graph representing the GPC matrix in Eq. (5.2).

the linear dependency of the code. Since the dummy symbol does not transmitted, the code rate does not change.

The factor graph representing the GPC matrix is illustrated by Fig. 5.2. Both factor graphs 5.1 and 5.2 represent the same code only over the first N symbols. Thus, this transformation of a factor graph based on the parity-check matrix transformation is same as the transformation described in Chapter 4 in the sense that it does not change the code itself.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.2}$$

For a GPC matrix, let the first N columns correspond the codeword symbols, and let other columns correspond dummy symbols. The receiver lets the received symbols corresponding to dummy symbols be erasure. In this chapter, we explain a transformation of a factor graph by using procedures of a transformation from an original parity-check matrix to a GPC matrix.

5.5 Decoding Methods for LDPC codes on the BEC

This section describes two decoding methods of LDPC codes on the BEC. One is a message-passing decoding [18] on factor graphs, which is the BEC version of the sum-product decoding. The other is a decoding method based on updating a parity-check matrix. We will use the latter for the proof of theorems. In fact, these are similar algorithms.

We describe the procedure when using an M by N parity check matrix H, and then let the range of n be $1 \leq n \leq N$ and let the range of m be $1 \leq m \leq M$. When an M' by N' GPC matrix A is used, replace the range of n into $1 \leq n \leq N'$ and replace the range of m into $1 \leq m \leq M'$.

In this chapter, a superscript (t) means the number of iteration on a decoding method.

5.5.1 Decoding method I : Message-passing

We describe the message passing decoding using the similar notation for the sum-product decoding of LDPC codes. All messages and variables take values in $\{0, 1, ?\}$.

Step 0.
Initialize $\hat{x}_n^{(0)} = y_n$ for all n and set $t = 1$.

Step 1.
Compute all pair of (m, n) satisfying $H_{mn} = 1$:

$$q_{nm} = \hat{x}_n^{(t-1)}. \tag{5.3}$$

Step 2.
Compute all pair of (m, n) satisfying $H_{mn} = 1$:

$$r_{mn} = \begin{cases} \sum_{n' \in \{N(m) \setminus n\}} q_{n'm} \mod 2, & q_{n'm} \neq ? \text{ for all } n'; \\ ?, & otherwise. \end{cases} \tag{5.4}$$

Step 3.
Compute for all n :
if there is $m \in M(n)$ such that $r_{mn} \neq ?$,

$$\hat{x}_n^{(t)} = r_{mn}, \tag{5.5}$$

else then

$$\hat{x}_n^{(t)} = ?. \tag{5.6}$$

Step 4.
If $\hat{x}_n^{(t)} = \hat{x}_n^{(t-1)}$ for all n, then the algorithm halts.
Setting $t = t + 1$ and continue Step 1 until $\hat{x}_n^{(t)}$ is not ? for all n.

For the message passing, the following lemma is satisfied [6]

Lemma 1. *Assume that we use a LDPC code to transmit over the BEC and that we decode the received word in a message-passing decoding algorithm until either the codeword has been recovered or until the decoder fails to progress further. Let denote χ the subset of the set of variable nodes which is erased by the channel. Then the set of erased symbols which remain when the decoder stops is equal to the unique maximal stopping set of χ.*

For example, the message-passing decoding for the LDPC code with the parity-check matrix in Eq. (5.1) cannot correct all erasures when $\mathbf{y} = ???? * *$, where $*$ denotes an arbitrary symbol in $\{0, 1, ?\}$.

5.5.2 Decoding method II : Matrix-updating

We propose a decoding method is based on matrix-updating searching for rows that have only one symbol "1". This is similar as the peeling decoder [18][26]. Let $A[i]$ denote i-th column of a matrix A. $A[i] = 0$ means that all the elements of i-th column of a matrix A are zero.

Let $E(x)$ be the erasure support set of the sequence x; that is, $E(x) = \{i \mid x_i = ?\}$. Let $\eta^{(t)}$ denote the updated matrix in the t-th iteration with the same number of columns and rows as the parity-check matrix.

Step 0.

Initialize $\hat{x}_n^{(0)} = y_n$ for all n, and set $t = 1$.

Step 1.

For all n, set columns of η as follows:

$$\eta^{(t)}[n] = \begin{cases} H[n], & n \in E(\hat{x}^{(t-1)}); \\ 0, & otherwise. \end{cases} \tag{5.7}$$

If $|E(\hat{x}^{(t-1)})| = 0$ or all rows of $\eta^{(t)}$ have two or more symbol "1" then this algorithm halts.

Step 2.

For all m such that m-th row of $\eta^{(t)}$ has only one symbol "1" in n-th column,

$$\hat{x}_n^{(t)} = \sum_{n' \in N(m) \setminus n} \hat{x}_{n'}^{(t-1)} \mod 2. \tag{5.8}$$

For no-updating $\hat{x}_{n'}^{(t)}$, $\hat{x}_{n'}^{(t)} = \hat{x}_{n'}^{(t-1)}$.

Step 3.

Set $t = t + 1$ and go Step 1.

Comparing procedures of each steps of the message-passing decoding and the matrix-updating decoding, one can confirm these decoding methods are equivalent. Note that all rows of $\eta^{(t)}$ have two or more symbol "1" in the matrix-updating decoding means that corresponding $E(\hat{x}^{(t-1)})$ is a stopping set.

The following is an instance of the matrix-updating decoding with the code with the

parity-check matrix (5.1). Given $\mathbf{y} = \hat{\mathbf{x}}^{(\mathbf{0})}$ =??0000, we have

$$\eta^{(1)} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{5.9}$$

The second and third rows have only one symbol "1", and then decide $\hat{x}_1^{(1)}$ and $\hat{x}_2^{(1)}$ as following:

$$\hat{x}_1^{(1)} = (\hat{x}_3^{(0)} + \hat{x}_5^{(0)}) \mod 2, \tag{5.10}$$

$$\hat{x}_2^{(1)} = (\hat{x}_3^{(0)} + \hat{x}_6^{(0)}) \mod 2. \tag{5.11}$$

If $\hat{x}^{(t-1)}$ =????00, we have

$$\eta^{(t)} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}. \tag{5.12}$$

There are not rows with only one symbol "1". Hence the method cannot correct new symbols any more.

5.6 A Transformation of a Parity-check Matrix

We propose a transformation of a parity-check matrix H into a GPC matrix in this section and proof that there are erasure patterns such that the above decoding methods using the GPC matrix can correct but that using the original matrix cannot correct.

5.6.1 Transformation Procedure

Select a k by l submatrix P of H that satisfies the following conditions: any $k-1$ rows of P are linear independent, and k rows of P are linear dependent. A combination of any columns and any rows of H satisfying the condition can be selected as P by applying appropriate row and column permutations to H. Let P^c be the sequence of column indices of H corresponding to each column of P, and let P^r be that corresponding to each row of P. The elements in the sequence are in ascending order.

Let O be the M by k all zero matrix. Let I be the k by k identify matrix. Let Q be a k by N matrix that is defined as follows:

$$Q[i] = \begin{cases} P[j], & i \in P^c, P_j^c = i; \\ 0, & otherwise. \end{cases} \tag{5.13}$$

Let C be an 1 by $N+k$ matrix in which the all elements at first N columns are zero, and the others are one.

Construct an $(M+k+1)$ by $(N+k)$ matrix A' defined as follows:

$$A' := \left[\begin{array}{c|c} H & O \\ \hline Q & I \\ \hline \multicolumn{2}{c}{C} \end{array}\right]. \tag{5.14}$$

Note that no new linear constraints are applied to the code. From the first row to M-th row of A' (that correspond to H and O) is equivalent to H. From the $(M+1)$-th row to $(M+k)$-th row of A' (that correspond to Q and I) mean that dummy symbols are uniquely determined given codeword symbols. The last row of A' (that corresponds to C) means that all rows of P are linear dependent. Then, A' is a GPC matrix of H of which the first to N-th columns correspond to codeword symbols, and the $(N+1)$th to $(N+k)$th column correspond to dummy symbols.

Construct A applying the following row operations to A': add $(M+i)$th row to i-th row of for all $i(0 \leq i \leq k-1)$. A is also GPC matrix of H.

We apply this procedure to the parity-check matrix in Eq. (5.1) for the submatrix

$$P = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \tag{5.15}$$

which corresponds to the first 3 rows and 3 columns of H. Then $P^c = \{1,2,3\}$, and $P^r = \{1,2,3\}$. We have

$$A' = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}, \tag{5.16}$$

and

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{5.17}$$

Consider the number of non-zero elements in the GPC matrix A. Let the number of non-zero elements in P be p. The number of non-zero elements increases $2k+p$ for the

transformation procedure from H to A‘. The number of non-zero elements decreases p (corresponding to the operation for submatrix Q) and increases k (corresponding to the operation for submatrix I) for the transformation procedure from A' to A. Then number of non-zero elements increases $3k$ overall. If H is a sparse matrix, then A is also a sparse matrix.

5.6.2 Decoding using the GPC matrix

The decoding as stated the last section using the GPC matrix A satisfy the following theorems. We proof these theorems using the matrix-updating method.

Theorem 1. *There are no erasure patterns such that the decoding methods with an original matrix H can correct but the decoding methods with the GPC matrix A cannot correct.*

Proof:

Let e denote the sequence with length $N' - N$ of which all symbols are erasure. Let $z \in \{0, 1, ?\}^N$ be a sequence with length N. We can get the corresponding updated matrix η of H from $E(z)$. Consider a sequence z such that the h-th row of η where $h \in P^r$ has only one non-zero symbol. Let the position of the non-zero symbol denote n. Such a sequence z corresponds to the received sequence y or the temporary sequence $\hat{x}^{(t)}$ that the matrix-updating decoding can correct the symbol x_n using the h-th row of H. It is only necessary for the proof that we show the matrix-updating method using the GPC matrix A can correct the symbol x_n given the sequence z' with length N', where z' is the combined sequence of z and e. We can get the corresponding updated matrix η' of A from $E(z')$.

First, consider the case of $n \in P^c$. By the procedure of the transformation from A' to A, $A_{hi} = 0$ for all $i \in P^c$. Then the h-th row of η' has only one non-zero element at a column corresponding to a dummy symbol. The $(h + M)$-th row of η' has two non-zero elements at both the n-th column and a column corresponding to a dummy symbol. Hence, the matrix-updating method can correct the dummy symbol using the h-th row of A and can correct the symbol z_n using the $(h+M)$-th row of A at the next iteration. Next, consider the case of $n \notin P^c$. By (5.13) and (5.14), $A_{(h+M)j} = 0$ for all $j \notin P^c$. Therefore the $(h + M)$-th row of η' has only one non-zero element at a column corresponding to a dummy symbol. The h-th row of η' has two non-zero elements at both the n-th column and a column corresponding to a dummy symbol. Hence, the matrix-updating method can correct the dummy symbol using the $(h + M)$-th row of A and can correct the symbol z_n using the h-th row of A at the next iteration. □

Theorem 2. *Consider a GPC matrix A transformed based on a code with a parity-check matrix H and the sub-matrix P. Let A^* be the sub-matrix of A corresponding to the all columns $n \in S$ and the all rows $m \in P^r$. If the following conditions are satisfied, the decoding methods using A can correct a erasure that cannot be corrected using the original matrix H.*

- All codeword symbol x_i such that $i \in P^c$ are contained in a stopping set S of the code.

- There is only one non-zero element in the submatrix A^*.

Proof:

We denote the position of the non-zero element on the second condition by n' and m'. Consider a stopping set S and a sequence z of which elements are denoted by

$$z_n = \begin{cases} ?, & n \in S; \\ 0, & otherwise. \end{cases} \tag{5.18}$$

Note that the decoding method using the original parity-check matrix H cannot correct any symbols in the sequence z. Also in this proof, let z' be the combined sequence of z and e, and consider the corresponding updated matrix η' of A from $E(z')$. The m-th rows of η' has only one non-zero symbol for all $m \in \{P_r \setminus m'\}$. The non-zero symbols are in the columns of corresponding to dummy symbols. Then the decoding method using the GPC matrix A can correct the symbols x_n for all dummy symbols $(N+1) \leq n \leq (N+k)$ except only one dummy symbol; the dummy symbol corresponds the column with the non-zero element in the m'-th row. At the next iteration, the decoding method can correct the dummy symbol using the last row of A. Overall the decoding method can correct the symbol $x_{n'}$ using the m'-th row of A. □

For example, consider the submatrix P in Eq. (5.15) and the GPC matrix A in Eq. (5.17). The sub-matrix A^* given by that $m \in \{1,2,3\}$ and $n \in \{1,2,3,4\}$ has only one element at A_{14}. Then it satisfies the conditions.

The factor graph representing the GPC matrix in Eq. (5.17) is illustrated in Fig. 5.3. We present an instance of the matrix updating method using the GPC matrix (5.17) in the following. Let $E^{(t)} = \{1,2,3,4,7,8,9\}$, and then we have

$$\eta^{(t)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{5.19}$$

We can correct x_8 and x_9 by using the second and third rows of $\eta^{(t)}$. We have $E^{(t+1)} = \{1,2,3,4,7\}$ and

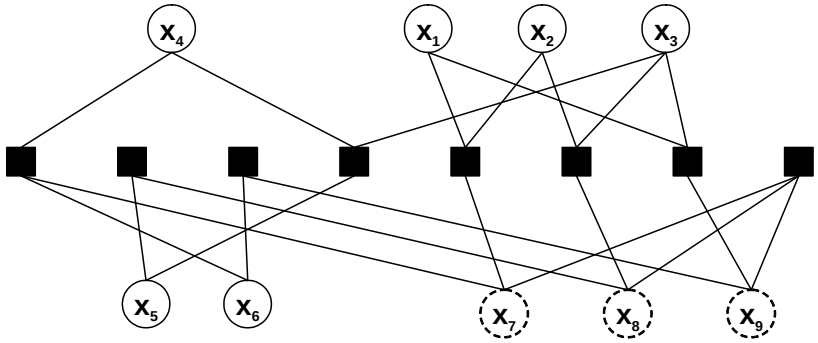

Fig. 5.3 Factor graph representing the GPC matrix in Eq. (5.17).

$$\eta^{(t+1)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{5.20}$$

We can correct x_7 by using the last row of $\eta^{(t+1)}$. We have $E^{(t+2)} = \{1, 2, 3, 4\}$ and

$$\eta^{(t+2)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{5.21}$$

We continue iterations, and then all codeword symbols are corrected.

5.7 Notes on the Proposed Transformation

5.7.1 ML Decoding and the Conditions in Theorem 2

The conditions in Theorem 2 means that the unknown variable $\hat{x}_{n'}$ can be solved by using all simultaneous equations corresponding to the m-th rows of H, where $m \in P^r$. For example, for $\mathbf{y} =$????00, the message-passing decoding cannot correct erasure any more. However, by summing the first row, the second and third one of H, we can have $\hat{x}_4 = \hat{x}_5 = 0$.

We say that our proposed transformation enables the message-passing algorithm to provide

Table. 5.1 Data of transformed matrices.

	N'	M'	R_i : average weight distribution of rows	#1
Ensemble 1	305.8	280.2	$R_3 : 0.7607$ $R_4 : 0.1214$ $R_5 : 0.1071 R_6 : 0.0107$	943.8
Ensemble 2	300.2	290.2	$R_3 : 0.7448$ $R_4 : 0.0517$ $R_5 : 0.1206$ $R_6 : 0.0793$ $R_7 : 0.0034$	1026.2

the solution. Therefore, the message-passing decoding approaches the ML decoding by using our proposed transformation.

5.7.2 Applying Proposed Transformation to Small Cycles

We need to consider how the submatrix P is chosen. The factor graph that represented by the submatrix P includes a cycle because of the linear dependencies of P. For example, the factor graph that represented by Eq. (5.15) is a cycle with the length 6. Then, we can choose the submatrix P so that the factor graph representing P is a small cycle. A similarly transformation for cycles with the length 4 of our proposed transformation is proposed in [14].

5.8 Numerical Experiments

We evaluate the decoding performance using our proposed transformation.

Consider two (4,8)-LDPC code ensembles with $N = 128$, Ensemble 1 and Ensemble 2. Ensemble 1 is that cycles with the length 4 are contained in the factor graph. Ensemble 2 is that cycles with the length 4 are not contained in the factor graph. We randomly choose five parity-check matrices from each ensemble.

First, we find short stopping sets of each code as many as possible. We also choose submatrices that satisfy conditions of the submatrix P over columns of H corresponding each stopping set. For each chosen submatrix such that satisfies the conditions in Theorem 2, we apply our proposed transformation based on the submatrix. We apply the transformation as many as possible for short stopping sets.

Table 5.1 shows the following data averaged by the five transformed parity-check matrixes. N' and M' are column length and row length of a transformed matrix, respectively. #1 means the number of symbols "1" in a parity-check matrix. R_i means weight distribution of each rows of a parity-check matrix.

We compare decoding performance using a message-passing algorithm with a transformed matrix to that with the original parity-check matrix. For each matrix and an erasure probability, 2×10^7 codewords are transmitted. The symbol erasure rate (SER) shown in the Fig. 5.4 and Fig. 5.5.

5.9 Discussion

The result for the Ensemble 1 shows that we can improve the performance for LDPC codes such that its factor graph contains many cycles with the length 4 by our proposed

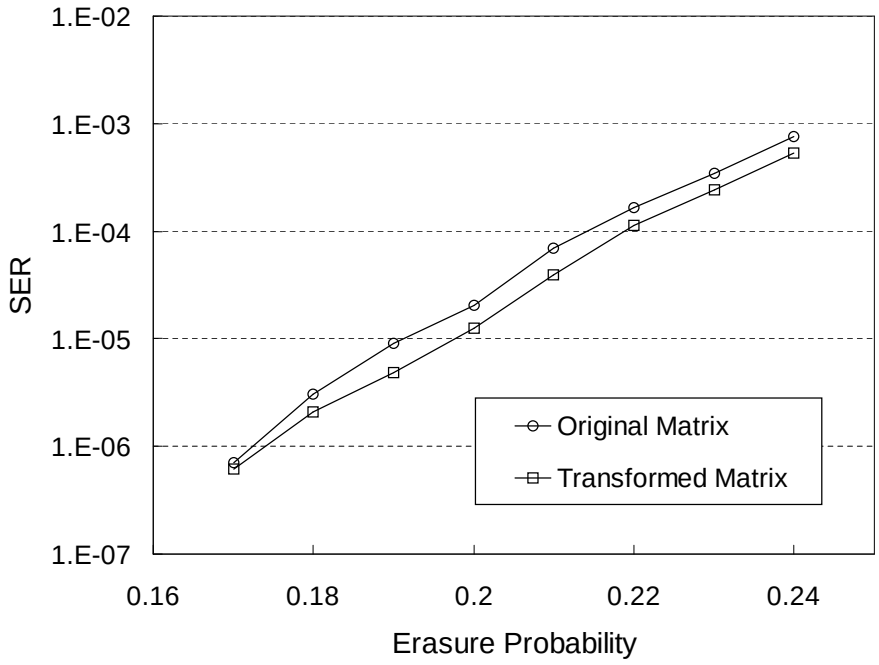

Fig. 5.4 Result for Ensemble 1.

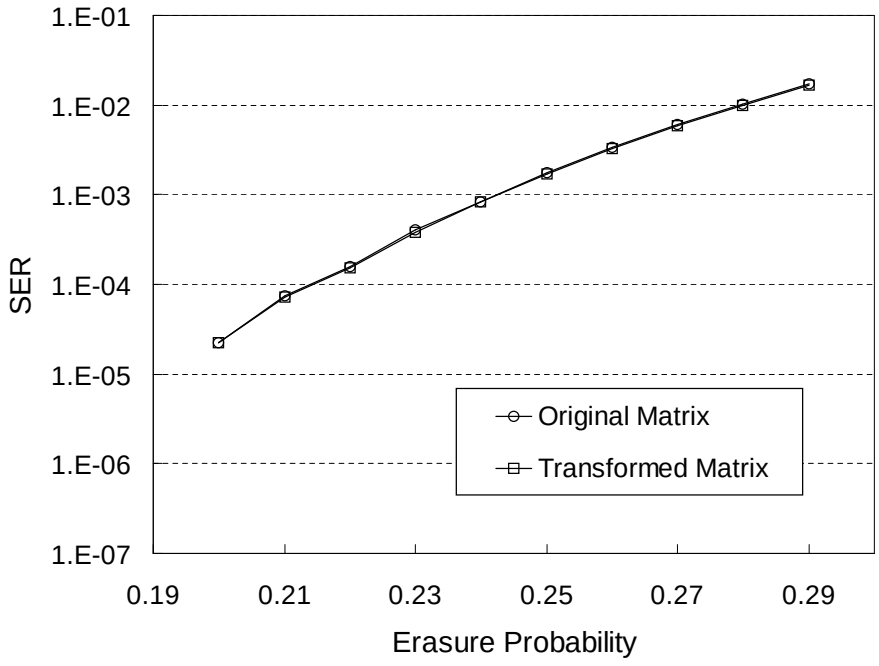

Fig. 5.5 Result for Ensemble 2.

transformation. It also shows that columns in H of randomly chosen LDPC codes that contain cycles with the length 4 are apt to be contained in a stopping set.

On this other hand, The result for the Ensemble 2 shows that we cannot improve the performance for LDPC codes such that its factor graph does not contain many cycles with the length 4 by our proposed transformation.

These results are same to the result according to the clustering method on the BIAWGN channel in Chapter 4. In both cases, the performance of LDPC codes with cycles with the length 4 can be improved by transformations of the factor graph.

5.10 Concluding Remarks

In Chapter 4 and Chapter 5, we focused on the transformation of factor graphs.

In this chapter, we proposed a new transform method of factor graph representing LDPC codes and showed that there are channel erasure patterns that can be corrected by the message-passing decoding over the transformed factor graph and that cannot correct by the message-passing decoding over the original factor graph. We proved the condition of such channel erasure patterns.

Moreover, we showed a relation between the transformation and the subgraphs corresponding to small cycles. By carrying out numerical experiments, we had verified that our method can improve the performance of the sum-product decoding with LDPC codes having small cycles on the BEC.

Chapter 6

Conclusion

Error-correcting codes on graphs have attracted attention over recent years in the study of error-correcting codes and have some interesting characteristics different from error-correcting codes based on the algebraic theory.

The factor graph not only represents both the code and the channel but also defines the procedure of decoding. When we design a code using the sum-product decoding on a channel, we must consider not only the performance of the code itself but also the performance of the sum-product decoding over the graph representing both the code and the channel.

For turbo codes and LDPC codes, it is shown that they have a good distance relation between pairs of codewords. In addition, they show good performance using the sum-product decoding on commonly used channels and the complexity of the decoding is low. Therefore, we say they not only are good codes themselves but also have an appropriate graph representation for the sum-product decoding.

In this thesis, we focused our discussion on their graph representations of turbo codes and LDPC codes and proposed new encoding and decoding schemes based on graph representations.

In Chapter 3, we considered a repeat request scheme using a feedback channel proposed by Shea. We extended this scheme by focusing on the characteristics of parallel concatenated codes. In addition, we proposed an adaptive encoding method based on the characteristics of both turbo codes and turbo decoding. We showed that the complexity introduced by the adaptive feature is low and demonstrated the performance of our proposed method by carrying out numerical experiments Our proposed method utilizes the characteristics of graph representation of codes.

Another characteristic is that a code can be represented by several factor graph representations. Namely, a code and a factor graph is not one-to-one relation. We can transform a factor graph representation to another one by some method. Chapter 4 and chapter 5 focused on the transformation of factor graphs.

In Chapter 4, we used the clustering method proposed by Pearl in the study of artificial intelligence. The clustering method was proposed to transform a non-tree structured graph representing a probabilistic model to a tree structured one. We used transformation methods to improve the decoding performance of LDPC codes with small cycles. We proposed a scheme of the sum-product decoding using the factor graph transformed by the clustering

method.

In Chapter 5, we proposed a new transform method of a factor graph to improve the performance of the sum-product decoding. By using the characteristics of the stopping sets, we showed conditions of channel erasure patterns which can be corrected by performing the sum-product decoding over the transformed factor graph but cannot be corrected by performing the sum-product decoding over the original factor graph.

Considering these transformations of a factor graph, we approach the problem from a point of view of designing the decoder without changing the code itself.

In the early parts of this thesis, we described encoding schemes. On the other hand, in the later parts of this thesis, we proposed decoding schemes. We can consider them at the same time, i.e. our proposed transformation method applies to positions which are wanted to provide high error-correcting. As a future work, we will consider such a combination of encoding and decoding using their graph representation of codes on graphs.

Acknowledgements

I would like to express my hearty gratitude to my supervisor, Professor Toshiyasu Matsushima, for his excellent wisdom and many pieces of advice on my study, academic activity and so on and so forth. I chose his laboratory to study and obtain knowledge, but in fact, the most valuable thing that I obtained from him was his logical though process. One year later I graduated from the university, I wanted to come back to his laboratory to study with him. I express my appreciation and will never forget his kindly agreement to accept it.

I would also like to express my gratitude to my co-advisors, Professor Shin'ich Oishi and Professor Shigeichi Hirasawa for helpful suggestions and invaluable advices during the peer-review process. I would especially like to thank Professor Shigeichi Hirasawa for his invaluable comments on my study.

I sincerely acknowledge Dr. Ryo Nomura at Aoyama Gakuin University and Dr. Kazuhiko Minematsu at NEC Corporation for their encouragement and advices.

I wish to thank my colleagues, Professor Yoshifumi Ukita at Yokobama College of Commerce, Dr. Yasunari Maeda at Kitami Institute of Technology, Takahiro Yoshida at Chuo University, Tota Suko, Shunsuke Horii at Waseda University, and Tomohiko Saito at Aoyama Gakuin University for worthwhile and delightful discussion with me. I would also like to thank all members of Hirasawa Laboratory at Waseda University.

Finally but not least, I would like to express my gratitude to my family for their understanding and support during my life. I would especially like to express my sincere gratitude to my parents.

March 2009.
Naoto Kobayashi.

Bibliography

[1] S. M. Aji and R. J. McEliece, "The Generalized Distributive Law," IEEE Trans. Inf. Theory, vol. 46, pp. 325-343, Mar. 2000.

[2] L. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans. Inf. Theory, vol. 20, pp. 284-287, Mar. 1974.

[3] S. Benedetto and G. Montorsi, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes," IEEE Trans. Inf. Theory, vol. 42, pp. 409-428, Mar. 1996.

[4] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in Proc. IEEE Int. Conf. Commun., vol. 2, pp.1064-1070, May 1993.

[5] S. T. Brink, "Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes," IEEE Trans. Comm., vol. 49, pp. 1727-1737, Oct. 2001.

[6] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson and R. L. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," IEEE Trans. Inf. Theory, vol. 48, pp. 1570-1579, Jun. 2002.

[7] S. Dolinar and D. Divsalar, "Weight Disributions for Turbo Codes using Random and Nonrandom Permutations," TDA Progress Report, Jet Propulsion Lab., pp. 45-122, Aug. 1995.

[8] W. Feng, J. Yuan and B. S. Vucetic, "A Code-matched Interleaver Design for Turbo Codes," IEEE Trans. on Commun., vol. 50, pp. 926-937, Jun. 2002.

[9] R. G. Gallager, "Low-density parity-check codes," IRE Trans. Inf. Theory, vol.8, pp. 21-28, Jan. 1962.

[10] R. G. Gallager, Information Theory and Reliable Communication, New York: Wiley, 1968.

[11] W. J. Van Gils, "Two topics on linear unequal error protection codes: Bounds on their length and cyclic code classes, " IEEE Trans. Inf. Theory, vol. 29, pp. 866-876, Nov. 1983.

[12] R. W. Hamming, "Error detecting and error-correcting codes," The Bell System Technical Journal, vol. 29, pp. 147-160, 1950.

[13] X. Y. Hu, E. Eleftheriou and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," IEEE Trans. Inf. Theory, vol. 51, pp. 386-398, Jan. 2005.

[14] K. Kasai, T. Shibuya and K. Sakaniwa, "A Code-Equivalent Transformation Removing Cycles of Length Four on Tanner Graphs (in Japanese)," Technical Report of IEICE,

IT2004-42, Sep. 2004.

[15] Y. Kou, S. Lin and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: arediscovery and new results," IEEE Trans. Inf. Theory, vol. 47, pp. 2711-2736, Nov. 2001.

[16] F. R. Kschischang, B. J. Frey and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," IEEE Trans. Inf. Theory, vol. 47, pp. 498-519, Feb. 2001.

[17] S. Lin and D. Costello, Jr., Error control coding: fundamentals and applications (second edition), Englewood Cliffs, NJ: Prentice-Hall, 2004.

[18] M. Luby, M. Mitzenmacher, M. Shokrollahi and D. Spielman, "Efficient erasure correcting codes," IEEE Trans. Inf. Theory, vol. 47, pp. 569-584, Feb. 2001.

[19] R. J. McEliece, D. J. C. MacKay and C. Jung-Fu, "Turbo decoding as an instance of Pearl's belief propagation algorithm," IEEE Journal on Selected Areas in Communications, vol. 16, pp. 140-152, Feb. 1998.

[20] D. J. C. Mackay, "Good error-correcting codes based on very sparse matrices," IEEE Trans. Inf. Theory, vol. 45, pp. 399-431, Mar. 1999.

[21] http://www.inference.phy.cam.ac.uk/mackay/codes/504.504.3.504

[22] G. Miller and D. Burshtein, "Bounds on the Maximum-Likelihood Decoding Error Probability of low-density parity-check codes," IEEE Trans. Inf. Theory, vol. 47, pp. 2696-2710, Nov. 2001.

[23] L. C. Perez, J. Seghers and D. J. Costello, "A Distance Spectrum Interpretation of Turbo Codes," IEEE Trans. Inf. Theory, vol. 42, pp. 1698-1709, Nov. 1996.

[24] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inferene, Morgan Kaufmann, 1988.

[25] T. J. Richardson and R. L. Urbanke, "The Capacity of LDPC codes under Message-Passing Decoding," IEEE Trans. Inf. Theory, vol. 47, pp. 599-618, Feb. 2001.

[26] T. Richardson and R. Urbanke, Modern Coding Theory, Cambridge University Press, 2008.

[27] A. Roongta and J. M. Shea, "Reliability-based hybrid ARQ using convolutional codes," in Proc. IEEE Int. Conf. on Communications, vol. 4, pp.2889-2893, May 2003.

[28] D. N. Rowitch and L. B. Milstein, "On the performance of hybrid FEC/ARQ systems using rate compatible punctured turbo (RCPT) codes," IEEE Trans. on Commun., vol. 48, pp. 948-959, Jun. 2000.

[29] C. E. Shannon, "A Mathematical Theory of Communication," The Bell System Technical Journal vol. 27, pp. 379-423, 623-656, 1948.

[30] J. M. Shea, "Reliability-based hybrid ARQ," IEE Electronics Letter, vol. 38, pp. 644-645, Jun. 2002.

[31] M. Sipser and D. A. Spielman, "Expander codes," IEEE Trans. Inf. Theory, vol. 42, pp. 1710-1722, Nov. 1996.

[32] K. Tsukamoto, T. Matsushima and S. Hirasawa, "A Study of the Decision of Control Parameters for Adaptive Automatic-Repeat Request Strategy," Electronics and Communications in Japan, Part 1, vol. 84, pp. 61-70, 2001.

[33] T. Wadayama, Low Density Parity- Check Codes and Its Decoding. Algorithm (in

Japanese), Triceps, Japan, 2002.

[34] J. S. Yedidia, J. Chen and M.Fossorier, "Generating Code Representations Suitable for Belief Propagation Decoding," in Proc. 40th Allerton Conference Commun., Control, and Computing, Monticello, IL, Oct. 2002.

Publications

N. Kobayashi, T. Matsushima and S. Hirasawa, "Transformation of a Parity-Check Matrix for a Message-passing Algorithm over the BEC," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E89-A, No. 5, pp. 1299-1306, 2006.

N. Kobayashi, D. Koizumi, T. Matsushima and S. Hirasawa, "A Note on Error Correction Schemes with a Feedback Channel," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E89-A, No. 10, pp. 2475-2480, 2006.

N. Kobayashi, T. Matsushima and S. Hirasawa, "A Note on a Decoding Algorithm of Codes on Graphs with Small Loops," IEEE ITSOC Information Theory Workshop 2005 on Coding and Complexity (ITW2005), Proceedings of the IEEE Information Theory Workshop 2005, IEEE, pp. 108-112, 28th/29th Aug. - 1st Sept., Rotorua, New Zealand, 2005.

N. Kobayashi, T. Matsushima and S. Hirasawa, "A Note on Error Correction Schemes using LDPC codes with a High-Capacity Feedback Channel," Proceedings of the International Symposium on Information Theory 2007, IEEE, pp.2381-2385,IEEE International Symposium on Information Theory 2007 (ISIT2007), 24th - 29th June 2007, Nice, France, 2007.

N. Kobayashi, T. Matsushima and S. Hirasawa, "An Accurate Density Evolution Analysis for a Finite-State Markov Channel". Proceedings of Pre-ICM International Convention on Mathematical Sciences," Pre-ICM International Convention on Mathematical Sciences, 18th -20th Dec. 2008, Delhi, India.

N. Kobayashi, T. Matsushima and S. Hirasawa, "An Accurate Density Evolution Analysis for a Finite-State Markov Channel," Proc. of Society of Information Theory and its Applications (SITA2008), pp. 510-515, 2008.

N. Kobayashi, T. Matsushima and S. Hirasawa, "Construction of Error Correcting Codes using plural LDPC codes and Interleaving for a Finite-State Markov Channel (in Japanese)," IEICE Technical Report, IT2008-29, pp. 55-60, 2008.

Y. Masui, N. Kobayashi and T. Matsushima, "A Note on a Construction of Error Correcting Codes for Channels with Memory (in Japanese)," Proc. of Society of Information Theory

and its Applications (SITA2006), pp. 1-4, 2006.

N. Kobayashi, T. Matsushima and S. Hirasawa, "A Note on Transmission Schemes with Unequal Error Protection Codes and a Feedback Channel," Proc. of Society of Information Theory and its Applications (SITA2006), pp. 815-818, 2006.

N. Kobayashi, D. Koizumi, T. Matsushima and S. Hirasawa, "A Study of Reliability Based Hybrid ARQ Schemes Using a Recursive Systematic Convolutional Code," IEICE Technical Report, IT2005-52, pp. 7-11, 2005.

N. Kobayashi, T. Matsushima and S. Hirasawa, "A Note on an Error Correcting System with a Feedback Channel (in Japanese)," Proc. of Society of Information Theory and its Applications (SITA2005), pp. 343-346, 2005.

K. Amemiya, N. Kobayashi and T. Matsushima "A Selective-Repeat ARQ Scheme with Finite Receiver Buffer Based on Statistical Decision Theory (in Japanese)," Proc. of Society of Information Theory and its Applications (SITA2005), pp. 757-760, 2005.