國 立 交 通 大 學

電信工程學系碩士班

碩 士 論 文

適用於低密度奇偶檢查迴旋碼之改良式差值動態排程解碼演算法

Improved Residual-Based Dynamic Scheduling for Decoding of Low-Density Parity-Check Convolutional Codes

研究生：吳牧諶

指導教授：王忠炫 博士

中 華 民 國 九 十 九 年 八 月

適用於低密度奇偶檢查迴旋碼之改良式差值動態排程解碼演算法

Improved Residual-Based Dynamic Scheduling for Decoding of Low-Density Parity-Check Convolutional Codes

研究生：吳牧諶　　　　Student: Mu-Chen Wu

指導教授：王忠炫　　　　Advisor: Chung-Hsuan Wang

國立交通大學

電信工程學系碩士班

碩士論文

A Thesis

Submitted to Department of Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Communication Engineering

August, 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年八月

適用於低密度奇偶檢查迴旋碼之改良式差值動態排程解碼演算法

研究生：吳牧諶　　　　　　　　指導教授：王忠炫 博士

國立交通大學

電信工程學系碩士班

摘　要

前人的研究顯示，具有分式奇偶檢查矩陣的低密度奇偶檢查迴旋碼因為其Tanner 圖裡有長度為 4 的循環，所以位元錯誤率很差。然而我們最近的研究發現，如果透過穿刺的概念將原來的 Tanner 圖轉換成具有較大的最小循環長的Tanner 圖，那些具有分式奇偶檢查矩陣的低密度奇偶檢查迴旋碼的位元錯誤率可以跟具有多項式奇偶檢查矩陣的低密度奇偶檢查迴旋碼的位元錯誤率一樣好，或者更好。由於以往對於穿刺的低密度奇偶檢查碼的解碼而言，排程法常被用來進一步改善位元錯誤率或是增加解碼的收斂速度，所以我們在現有的排程法中選擇表現較好的排程演算法（EDS 演算法）來解那些具有分式奇偶檢查矩陣的低密度奇偶檢查迴旋碼，看看是否可以得到更好的位元錯誤率。在這篇論文中，我們首先修改了 EDS 演算法中的差值方程式，使得解碼過程中的更新順序更洽當。此外，透過觀察我們發現，使用排程法來解碼可能會有不收斂或者是收斂到非最佳碼字的問題，有鑑於此，我們提出兩個方法，分別使用擾動和位元翻轉來解決問題。根據模擬結果，無論是對於具有分式校驗矩陣的低密度校驗迴旋碼還是對於具有多項式校驗矩陣的低密度校驗迴旋碼，我們提出的排程演算法的位元錯誤率都比一些現有排程演算法的位元錯誤率好。

Improved Residual-Based Dynamic Scheduling for Decoding of Low-Density Parity-Check Convolutional Codes

Student: Mu-Chen Wu Advisor: Chung-Hsuan Wang

Department of Communication Engineering

National Chiao Tung University

Abstract

Previous studies on low-density parity-check convolutional codes (LDPC-CCs) revealed that LDPC-CCs with rational parity-check matrices (RPCM) have poor bit-error-rate (BER) performances due to the existence of lenth-4 cycles in their Tanner graphs. In our recent work, we found that we can transform the original Tanner graph of an LDPC-CC with an RPCM into a new Tanner graph with larger girth based on the concept of puncturing such that the LDPC-CC can have a comparable or even better BER performance than those of LDPC-CCs with polynomial parity-check matrices (PPCMs). For the decoding of punctured LDPC codes, sequential schedules are usually used to improve BER performances or speed up the convergence of the decoding. We select the well-performed efficient dynamic scheduling (EDS) among the available sequential schedules to decode those LDPC-CCs with RPCMs in order to obtain better BER performances. In this thesis, we firstly modify the residual function of EDS to have a more appropriate updating order. Besides, since several observations indicate that the decoding based on the original EDS or our improved EDS may not converge or converge to non-optimal codewords, two refined strategies based on the perturbation and the bit-flipping are hence proposed to mitigate these problems. Revealed by the simulations results, not only for RPCMs but also for PPCMs, our proposed algorithm can provide better BER performances than those of several existent schemes.

誌　謝

首先，我要感謝指導教授王忠炫博士兩年的教誨，也感謝實驗室翁健家學長和張力仁學長不論在學習研究上或生活上給予我的幫助，另外還要感謝實驗室同學和學妹的扶持和鼓勵，讓我度過了充實的兩年研究生的生活。最後，對一直在背後默默支持我的家人以及陪在我身旁的朋友們，我想說聲我愛你們。我由衷地感謝大家。

民國九十九年八月

研究生吳牧諶謹識於交通大學

Contents

List of Figures

Chapter 1

Introduction

In 1990s, low-density parity-check (LDPC) codes were rediscovered. LDPC codes were shown that they can achieve near-Shannon-limit performances with iterative message-passing decoding and sufficiently long block length. In 1999, Jiménez Felström and Zigangirov proposed low-density parity-check convolutional codes (LDPC-CCs) [1], which can be considered as convolutional counterparts of LDPC block codes. They showed that LDPC-CCs have comparable performances to those of LDPC block codes (LDPC-BCs). Furthermore, an LDPC-CC can be easily encoded in a systematic way only by adders and shift registers. It can also be encoded with arbitrary length of data bits.

Previous studies of LDPC-CCs revealed that LDPC-CCs with rational parity-check matrices have poor bit-error-rate (BER) performances due to the existence of length-4 cycles in their Tanner graphs. To acquire a better performance by the sum-product algorithm, in our recent work, we propose a procedure [6] based on the concept of puncturing for obtaining an equivalent Tanner graph with larger girth. To further enhance the BER performance and simultaneously accelerate the speed of convergence, many researchers suggest that the sequential schedules should be applied for decoding. For sequential schedules, they can be partitioned into two classes–deterministic scheduling and dynamic scheduling, where the former decides its updating order before the decoding while the latter continuously regulates its updating order during the decoding [3][4][5]. In the previous research, dynamic scheduling is shown to have a better performance than that of deterministic scheduling. Among many sequential schedules, we apply one well-performed dynamic sequential schedule, which is named efficient dynamic scheduling (EDS) [2], to decode LDPC-CCs.

In this thesis, we first modify the residual function of EDS to have a more appropriate updating order. The original residual function aims to speed up the convergence of the decoding while our improved residual function not only aims to speed up the convergence of the decoding but also consider which variable nodes can help other variable nodes. Moreover, we find that the decoding based on the original EDS or our improved EDS may not converge or converge to non-optimal codewords. Two refined strategies based on the perturbation and the bit-flipping are hence proposed to solve these problems. The former adds small noise to the received sequence to help the decoding process escape from the decoding trap while the later searches several codeword candidates around the original decoded codeword to obtain a better decoded result. Revealed by the simulations results, not only for rational parity-check matrices but also for polynomial parity-check matrices, our proposed algorithm can provide a better BER performance than those of several existent schemes.

The rest of this thesis is organized as follows: First of all, in Section 2 and 3, LDPC-BCs and LDPC-CCs are briefly described. The proposed algorithm and the simulation results are given in Section 4 and 5. In the end, the work is concluded in Section 6.

Chapter 2

Overview of Low-Density Parity-Check Codes

In this chapter, we first give a review of LDPC codes and Tanner graphs. Then the sum-product algorithm for the decoding of LDPC codes is also described.

2.1 LDPC Codes and Tanner Graphs

An LDPC code is a linear binary block code whose parity-check matrix $\mathbf{H}$ has low density of ones. If there are J 1's in every column and K 1's in every row, and the number of 1's in common between any two columns is smaller than 2, it is called a (J, K) regular LDPC code with the column weight $w_c = J$ and the row weight $w_r = K$. However, if the column weight or the row weight of an LDPC code is not constant, the LDPC code is called an irregular LDPC code.

An LDPC code is usually described by its Tanner graph. A Tanner graph is a bipartite graph and can be partitioned into two classes–variable nodes v_i's and check nodes c_j's, which represent the codeword bits and the check equations, respectively. In a Tanner graph, there is no edge connecting two nodes from the same class. If and only if the bit is included in the parity check, there is an edge connecting a variable node and a check node. The neighbors of one node are the nodes which connect to it. $N(v_i)$ and $N(c_j)$ denote the neighbors of the variable node v_i and the check node c_j, respectively.

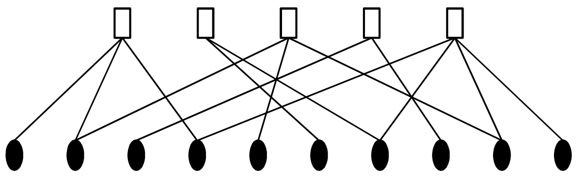


Figure 2.1: The Tanner graph of a rate $R = 1/2$ LDPC code.

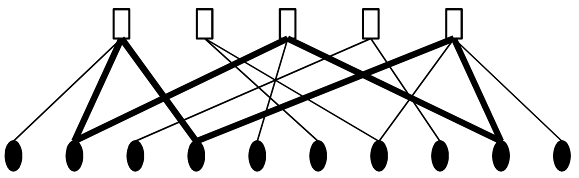


Figure 2.2: A cycle of length 6.

Example 2.1 *Consider a rate* $R = 1/2$ *irregular LDPC code with*

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

Its Tanner graph is composed of 10 *variable nodes and* 5 *check nodes, as shown in Fig. 2.1, where white squares represent check nodes and black circles represent variable nodes.* □

In a Tanner graph, if one can start from one node and go back to the same node through l edges with the condition that no node is passed through more than twice, there is a length-l cycle. Relatively speaking, in a parity-check matrix $\mathbf{H}$, if one can start from some "1" , sequentially walk vertically to another "1" and walk horizontally to the other "1", and walk back to the original "1" through l steps, there is a length-l cycle. A length-6 cycle in the Tanner graph of Ex. 2.1 is shown in Fig. 2.2. The girth of a Tanner graph is the minimum length of all cycles. The girth of the LDPC code in Ex. 2.1 is 6.

2.2 Sum-Product Algorithm

The sum-product algorithm (SPA) is an iterative decoding algorithm which is often used to decode LDPC codes. It can be operated on a Tanner graph that variable nodes and check nodes exchange extrinsic information iteratively. For SPA, it assumes that the received messages of every node in a Tanner graph are independent. If a Tanner graph is cycle-free, SPA is optimal for the decoding of corresponding LDPC code. However, short cycles in a Tanner graph increase the dependence of the messages and worsen the BER performance of corresponding LDPC code. SPA is also called flooding because all check nodes or all variable nodes are processed at the same time.

Assume transmitted bit x_i is priori equally likely to be $+1$ or -1 under binary phase shift keying (BPSK) scheme, thus the log a posterior probability (log-APP) ratio based on the channel output y_i is

$$X_{v_i} = log\left(\frac{P(x_i = +1|y_i)}{P(x_i = -1|y_i)}\right) = 2y_i/\sigma^2.$$

In the beginning of the algorithm, we initialize the message passed from variable node v_i to a check node c_j as $m^{(0)}_{v_i \to c_j} = X_{v_i}$, and the message passed from check node $c_j$to variable node v_i as $m^{(0)}_{c_j \to v_i} = 0$. In the lth iteration of the algorithm, firstly, each check node c_j computes the message

$$m^{(l)}_{c_j \to v_i} = 2tanh^{-1}\left(\prod_{v_k \in N(c_j)\backslash v_i} tanh\left(\frac{1}{2}m^{(l-1)}_{v_k \to c_j}\right)\right)$$

and sends it to its neighbors $N(c_j)$. On the other hand, each variable node v_i computes the message

$$m^{(l)}_{v_i \to c_j} = \sum_{c_k \in N(v_i)\backslash c_j} m^{(l)}_{c_k \to v_i} + X_{v_i}$$

and sends it to its neighbors $N(v_i)$. Secondly, we compute the log-APP ratio

$$Q_i = X_{v_i} + \sum_{c_j \in N(v_i)} m^{(l)}_{c_j \to v_i},$$

and make a hard decision

$$\hat{x}_i = \begin{cases} +1, & if Q_i > 0 \\ -1, & otherwise \end{cases}$$

for each variable node v_i. If all parity checks are satisfied, $\hat{\mathbf{x}}\mathbf{H}^T = \mathbf{0}$, or the maximum number of iterations is reached, then stop; otherwise, continue the algorithm.

Chapter 3

LDPC Convolutional Codes

3.1 Definition

Let

$$\mathbf{u}_{[0,t]} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_t),$$

where $\mathbf{u}_i = (u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(k)})$ and $u_i^{(\cdot)} \in GF(2)$, be the information sequence and

$$\mathbf{v}_{[0,t]} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_t),$$

where $\mathbf{v}_i = (v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(n)})$ and $v_i^{(\cdot)} \in GF(2)$, be the encoded sequence. A time-invariant LDPC-CC is defined as the set of all sequences $\mathbf{v}_{[0,\infty]}$ satisfying the equation $\mathbf{v}_{[0,\infty]}\mathbf{H}^T_{[0,\infty]} = \mathbf{0}$, where

$$\mathbf{H}^T_{[0,\infty]} = \begin{bmatrix} \mathbf{H}_0^T & \cdots & \mathbf{H}_{m_s}^T & & \mathbf{0} & \\ & \ddots & & \ddots & & \\ \mathbf{0} & & \mathbf{H}_0^T & \cdots & \mathbf{H}_{m_s}^T & \\ & & & \ddots & & \ddots \end{bmatrix}$$

is a semi-infinite transposed parity-check matrix, called *syndrome former*. For a rate $R = k/n$ code, the elements of $\mathbf{H}^T_{[0,\infty]}$ are submatrices of dimension $(n-k) \times n$ and m_s is the *syndrome former memory*.

The same with a convolutional code, information sequence $\mathbf{u}_{[0,\infty]}$ and encoded sequence $\mathbf{v}_{[0,\infty]}$ can be represented by polynomial vectors

$$\mathbf{U}(D) = (U_1(D), U_2(D), \ldots, U_k(D))$$
$$\mathbf{V}(D) = (V_1(D), V_2(D), \ldots, V_n(D)),$$

where $U_i(D) = u_0^{(i)} + u_1^{(i)}D + \cdots$ and $V_i(D) = v_0^{(i)} + v_1^{(i)}D + \cdots$, respectively. The parity check matrix can also be denoted by

$$\mathbf{H}(D) = \mathbf{H}_0 + \mathbf{H}_1 D + \cdots + \mathbf{H}_{m_s} D^{m_s},$$

and $\mathbf{V}(D)$ is a codeword if and only if $\mathbf{V}(D)\mathbf{H}^T(D) = \mathbf{0}$.

3.2 Encoding

We usually require $\mathbf{H}_0$ to be full rank in order to take this property to easily encode a LDPC-CC with only shift registers and adders.

Example 3.1 *Consider a LDPC-CC with $R = 1/3$, which can be specified by*

$$\mathbf{H}(D) = \begin{bmatrix} 1 & D & D^3 \\ D^3 & D^2 & 1 \end{bmatrix}.$$

First of all, we decompose it into a superposition of matrices in different degrees of delay

$$\begin{aligned} \mathbf{H}(D) &= \mathbf{H}_0 + \mathbf{H}_1 D + \ldots + \mathbf{H}_{m_s} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} D + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} D^2 + \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} D^3. \end{aligned}$$

Then, since $\mathbf{v}_{[0,\infty]}\mathbf{H}_{[0,\infty]}^T = \mathbf{0}$, we relate encoded bits of time t with past bits by the equation

$$\mathbf{v}_t \mathbf{H}_0^T + \mathbf{v}_{t-1}\mathbf{H}_1^T + \ldots + \mathbf{v}_{m_s}\mathbf{H}_{m_s}^T = \mathbf{0},$$

and obtain the following equations

$$\begin{cases} v_t^{(1)} + v_{t-1}^{(2)} + v_{t-3}^{(3)} = 0 \\ v_t^{(3)} + v_{t-2}^{(2)} + v_{t-3}^{(1)} = 0. \end{cases}$$

By setting $v_t^{(2)}$ be the information bit u_t, we can solve the simultaneous equations and generate the parity-check bits of time t. The encoder is shown in Fig. 3.1. Only 2 *adders and* 9 *shift registers are needed.* □

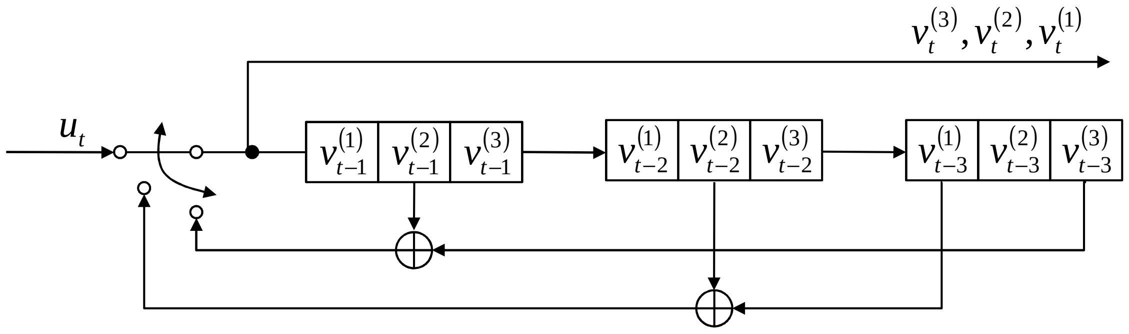


Figure 3.1: A rate $R = 1/3$ and syndrome former memory $m_s = 3$ LDPC-CC encoder

3.3 Decoding

Viterbi algorithm is rarely used to decode an LDPC-CC, because the syndrome former memory m_s of an LDPC-CC is usually large. However, since $\mathbf{H}$ is low-density, SPA is considered instead. Different from decoding an LDPC block code, it's hard to process whole codeword at one time since the codeword length can go to infinity. As being shown in the Fig. 3.2, there's a sliding window which stores the data under process. The Tanner graph in the Fig. 3.2 is derived from the $\mathbf{H}(D)$ in the Ex. 3.1. The window is composed of I, which equals to the iteration number, processors with size $(m_s + 1)$ time instants. Every time we receive n channel outputs, we put n variable nodes and $n - k$ check nodes into the window and pop out the last n variable nodes and $n - k$ check nodes from it. We firstly activate the front $n - k$ check nodes in each processor, and secondly activate the last n variable nodes in each processor. Once check nodes at time $t + 4I - 1$ are updated, all check nodes which connect to variable nodes at time $t + 4(I - 1)$ have already been updated once. Thus, all those variable nodes can compute $m_{v_i \to c_j}^{(1)}$ and complete their first iteration. In the other hand, when the check nodes at $t + 4(I - 1) - 1$ are updated, all variable nodes connect to them have already been updated once. Thus, all those check nodes can compute $m_{c_j \to v_i}^{(2)}$.

However, these processors are operated independently, because one variable node at time t only connects to check nodes which locate between time t to time $t + m_s$, and one check node at time $t + m_s$ only connects to variable nodes which locate between time t to time

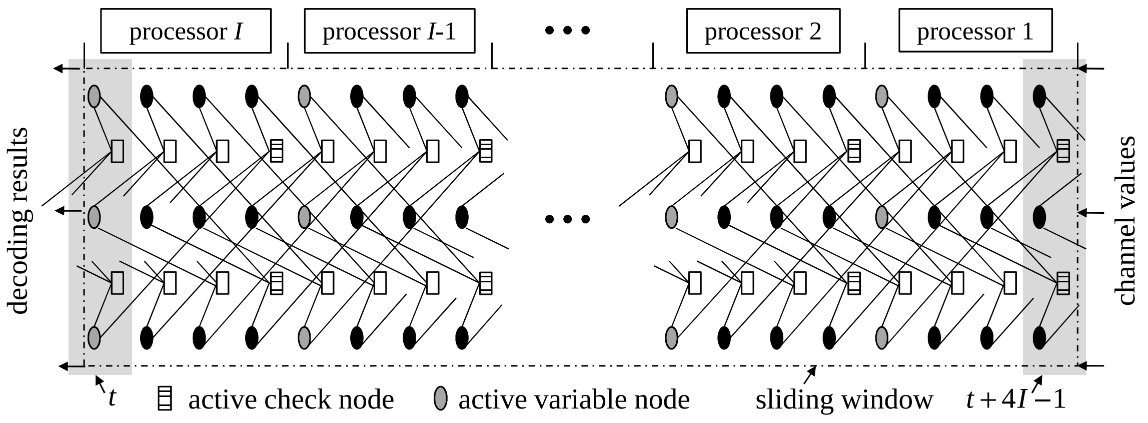


Figure 3.2: Decoding window of LDPC-CC

$t+m_s$. Thus active check nodes in some processor only access variable nodes located in their processor and vice versa. After those active nodes are updated, we put next n received bits into the window and pop out the last n bits again. Obviously, each variable node will be updated I times in the window. The complete flooding algorithm is described in Algorithm 1.

Algorithm 1 Flooding for LDPC-CC

```
 1: Pop in $n$ variable nodes and $n-k$ check nodes.
 2: for $i = 1$ to $I$ do
 3:     Activate the front $n-k$ check nodes in processor $i$.
 4:     Activate the last $n$ variable nodes in processor $i$.
 5: end for
 6: Pop out the last $n$ variable nodes and $n-k$ check nodes in the window.
 7: if termination criterion is satisfied then
 8:     Leave the algorithm.
 9: else
10:     Go back to step 1.
11: end if
```

3.4 Girth and Rational Parity-Check matrices

One can directly check whether small girth exists or not in an LDPC-CC by $\mathbf{H}(D)$. First of all, we introduce how to find a cycle in $\mathbf{H}(D)$. We start from some nonzero term D^{n_0} in $\mathbf{H}(D)$, walk along the column to another nonzero term D^{n_1}, walk along the row to the

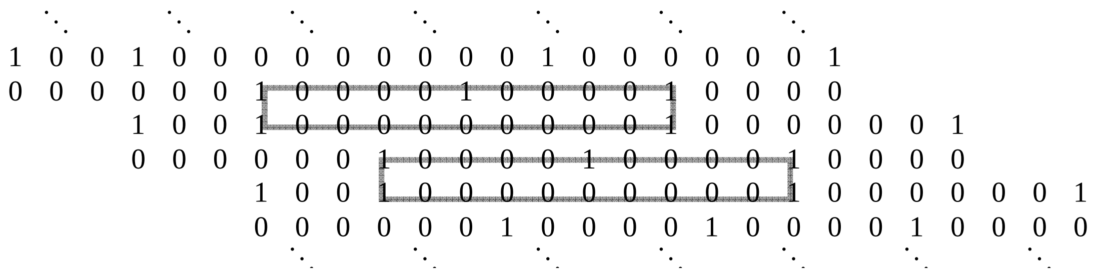


Figure 3.3: Cycles of length 4 in $\mathbf{H}$

other nonzero term D^{n_2}, and repeat these two steps sequentially until a cycle is found. Define $d(n_i, n_{i+1}) = n_{i+1} - n_i$ be the displacement of walking from D^{n_i} to $D^{n_{i+1}}$. A path $(D^{n_0}, D^{n_1}, \ldots, D^{n_{2l}}, D^{n_{2l+1}}, D^{n_0})$ forms a cycle of length $2l$ if and only if the summation of all vertical displacements is 0

$$D_v = d(n_0, n_1) + d(n_2, n_3) + \cdots + d(n_{2l}, n_{2l+1}) = 0.$$

Example 3.2 *Consider*

$$\mathbf{H}(D) = \begin{bmatrix} 1+D & D^4 & D^6 \\ D^2 & D^5 & D^3 \end{bmatrix}.$$

There's a path (D^2, D, D^4, D^5, D^2) forming a cycle of length 4 with $D_v = (2-1)+(4-5) = 0$, as shown in Figure 3.3. $\square$

For LDPC-CC, multinomial terms, excluding binomial terms, in $\mathbf{H}(D)$ promise girth less than or equal to 6. For example, we can easily find that a trinomial term $D^a + D^b + D^c$, where $a < b < c$, contains a cycle $(D^a, D^b, D^c, D^a, D^b, D^c, D^a)$ with length 6. Moreover, if $b-a = c-b$, the length of the cycle will be 4. Thus, in the past, only monomial and binomial terms were discussed. Besides, rational $\mathbf{H}(D)$ was also rarely discussed because small girth follows traditional Tanner graph, which was obtained by multiplying the denominator to the corresponding row or expanding the rational term into an infinite geometric series. Nevertheless, we can obtain a new Tanner graph with larger girth by using dummy variable nodes [6] to replace rational or multinomial terms.

Example 3.3 *Consider*

$$\mathbf{H}(D) = \begin{bmatrix} 1 & D & D^3 \\ D^3 & D^2 & \frac{1}{1+D} \end{bmatrix}.$$

If we multiply $(1+D)$ *to the second row of* $\mathbf{H}(D)$*, we can obtain*

$$\mathbf{H}^{'}(D) = \begin{bmatrix} 1 & D & D^3 \\ D^3+D^4 & D^2+D^3 & 1 \end{bmatrix}.$$

Unfortunately, there's cycle $(D^3, D^2, D^3, D^4, D^3)$ *with length* 4*. If we expand* $\frac{1}{1+D}$ *into* $1+D+D^2+D^3+\cdots$*, there's also a cycle* $(1, D, D^2, D, 1,)$ *with length* 4*. Both methods promise girth equal to* 4*. Nevertheless, let dummy variable node* $M(D) = \frac{1}{1+D}V_3(D)$ *and obtain*

$$\mathbf{H}^{''}(D) = \begin{bmatrix} 1 & D & D^3 & 0 \\ D^3 & D^2 & 0 & 1 \\ 0 & 0 & 1 & 1+D \end{bmatrix}.$$

One can easily check there's no cycle with length 4 *in* $\mathbf{H}^{''}$*.* □

Using dummy variable nodes equals using a super code with rate $R = k/(n+n')$ to replace the original one, where n' is number of dummy variable nodes. However, to maintain the same rate, previous researcher directly punctured the dummy variable nodes. In order to improve performance, we find which columns being punctured can acquire the best performance through simulations.

Chapter 4

The Proposed Algorithm

4.1 Residual-Based Dynamic Scheduling

For the decoding of LDPC-CCs, the flooding algorithm is usually used. However, there is one troublesome problem that the hardware complexity of the decoder is proportional to the number of iterations, which can be few hundreds. Thus, we hope to develop another decoding algorithm. Although the hardware complexity of the decoder is limited, the decoding algorithm still has a good performance. For LDPC-BCs, scheduling is one common way to accelerate the speed of convergence and improve the performance, especially for punctured ones. Since we originally focus on rational $\mathbf{H}(D)$, which is processed by dummy variable nodes and punctured to maintain the same code rate, scheduling is considered.

Sequential schedules can be partitioned into two classes–deterministic ones and dynamic ones. Deterministic sequential schedules decide their updating order before the decoding while dynamic sequential schedules continuously regulate their updating order during the decoding. However, dynamic sequential schedules have been shown that they have better BER performances and faster speed of convergence than deterministic sequential schedules. For dynamic sequential schedules, they can be further partitioned into several classes based on how to decide their updating orders. A residual-based dynamic sequential schedule defines a residual function and decides which variable node, check node or edge should be updated first based on its residual function. Residual belief propagation (RBP) [5], node-wise residual belief propagation (NWRBP) [5], and efficient dynamic scheduling (EDS) [2] are residual-based dynamic sequential schedules. For RBP, it defines its residual function

$$F^{(l)}(m_{c_j\to v_i}) = |m^{(l+1)}_{c_j\to v_i} - m^{(l)}_{c_j\to v_i}|$$

based on messages passed from check nodes to variable nodes and iteratively updates the edge with the largest residual. For NWRBP, it also defines its residual function

$$F^{(l)}(c_j) = \max_{\forall v_i \in N(c_j)} |m^{(l+1)}_{c_j\to v_i} - m^{(l)}_{c_j\to v_i}|$$

based on messages passed from check nodes to variable nodes. However, it not only updates the edge with the largest residual but also other edges which connect to the same check node. For EDS, it defines its residual function

$$F^{(l)}(v_i) = \frac{|Q_i^{(l)} - Q_i^{(l-1)}|}{|Q_i^{(l)} + Q_i^{(l-1)}|}$$

based on log-APP ratios of variable nodes and iteratively updates the variable node with the largest residual. In [2], EDS was shown that it has a better performance than those of RBP and NWRBP.

For EDS, the residual of variable node v_i is defined by

$$F^{(l)}(v_i) = \frac{|Q_i^{(l)} - Q_i^{(l-1)}|}{|Q_i^{(l)} + Q_i^{(l-1)}|}.$$

Every time it picks one variable node with the largest residual and updates the neighboring check nodes of the variable node. Neighbors of those check nodes are also updated. Then it computes new residuals for updated variable nodes and reset the residual of the selected one to 0. Now we apply EDS to decode LDPC-CCs. Different from the flooding algorithm, the window is composed of only one processor with size $K\times(m_s+1)$ time instants. Here, K does not promise that every node can be updated K times. For convenience, we partition the window into K blocks with size (m_s+1) time instants. Every time we receive $(m_s+1)\times n$ channel outputs, we put $(m_s+1)\times n$ rather than n variable nodes and $(m_s+1)\times(n-k)$ check nodes into the window, and pop out the last $(m_s+1)\times n$ variable nodes and $(m_s+1)\times(n-k)$ check nodes from it. We give those variable node, who were just put in the window, a initial residual $F^{(0)}v_i = X_{v_i}$ and sort all variable nodes' residuals. Then we pick the variable node with the largest residual, update its neighboring check nodes, and also update the neighbors of those check nodes. Afterwards, We compute the residuals for updated variable nodes and reset the residual of the selected one to 0. We call these operations hybrid operation PUC and repeat it N times. Then, we shift the variable nodes and the check nodes again.

For RBP and NWRBP, their residual were defined by

$$F^{(l)}(c_j \to v_i) = |m^{(l)}_{c_j \to v_i} - m^{(l)}_{c_j \to v_i}|$$

and

$$F^{(l)}(c_j) = \max_{\forall i \in N(c_j)} |m^{(l)}_{c_j \to v_i} - m^{(l)}_{c_j \to v_i}|$$

, respectively. Both two functions only describe the difference between a message before and after being updated. However, for EDS, when there are more than one variable nodes with the same $|Q_i^{(l)} - Q_i^{(l-1)}|$, it picks the variable node with the smallest reliability. It believes that updating variable nodes with small reliability first can accelerate the speed of convergence. Nevertheless, updating unreliable variable nodes first may influence other reliable variable nodes and even cause errors. Thus, we modify the residual function

$$F^{(l)}(v_i) = \frac{|Q_i^{(l)} - Q_i^{(l-1)}|}{|Q_i^{(l)}/Q_i^{(l-1)}|}.$$

For those variable nodes with the same $|Q_i^{(l)} - Q_i^{(l-1)}|$, we pick the one with the largest reliability. Besides, for a variable nodes whose sign is changed, we want to give it a higher residual. Thus, we further modify the residual function

$$F^{(l)}(v_i) = \frac{|Q_i^{(l)} - Q_i^{(l-1)}|}{|Q_i^{(l)}/Q_i^{(l-1)} + 1|}.$$

For variable nodes with the same $|Q_i^{(l)} - Q_i^{(l-1)}|/|Q_i^{(l)}/Q_i^{(l-1)}|$, we pick the one whose sign is changed. We call the algorithm with the modified residual function modified EDS, and it is completely described in Algorithm 2.

In modified EDS, every time we pick one variable node from the window and update the neighboring check nodes. However, the neighboring check nodes of the variable nodes

Algorithm 2 Modified EDS

```
1: Pop $(m_s + 1) \times n$ variable nodes and $(m_s + 1) \times (n - k)$ check nodes into the window.
2: Pop out the last $(m_s + 1) \times n$ variable nodes and $(m_s + 1) \times (n - k)$ check nodes and
   shift variable nodes and check nodes in the window to the left.
3: Set messages passed from these check node to 0.
4: Let $F^{(0)}v_i = X_{v_i}$.
5: Sort all variables' residuals.
6: for $i = 1$ to $N$ do
7:     Pick the variable node with the largest residual, update the neighboring check nodes,
       and update the neighbors of those check nodes.
8:     Compute the residuals of updated variable nodes.
9:     Set the residual of the selected variable node to 0.
10:    Sort all variables' residuals again.
11: end for
12: Go back to step 1.
```

in the last block may connect to the variable nodes which have already leave the window. On the other side, the neighboring check nodes of the variable nodes in the first block probably have not enter the window yet. Thus, we exclude the last and front $(m_s + 1) \times n$ variable node from the selection. Besides, instead of using a huge stack to sort all variable nodes' residuals, we use $K - 2$ stacks to sort the residuals of variable nodes in each block and another stack to sort the largest residual in each stack. One stack corresponds to one block. Thus, there is no need to clear stacks and sort all residuals again when new data enters. And that is why we propose popping in $(m_s + 1) \times n$ instead of n channel outputs each time. However, once the residual of a variable node is modified, there are $log((m_s + 1) \times n) + log(K - 2) \approx log((m_s + 1) \times n)$ comparisons.

For the flooding algorithm, it costs I processors and $I \times (m_s + 1) \times (n + (n - k) \times \sum w_r)$ storage elements. However, for modified EDS algorithm, it costs one powerful processors, $K \times (m_s + 1) \times (n + (n - k) \times \sum w_r)$ storage elements, $K - 2$ stacks of size $(m_s + 1) \times n$ and one stack of size $K - 2$. Additionally, the front needs a buffer of size n to hold new received values. Nevertheless, the buffer size of the latter should be expand to $(m_s + 1) \times n$.

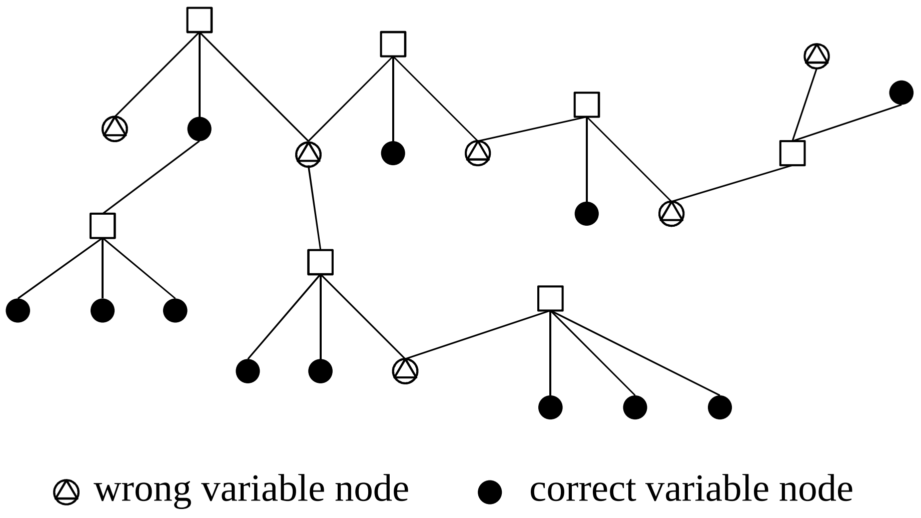


Figure 4.1: A subgraph that most check nodes connect to even wrong variable nodes.

4.2 Improved Scheme Based on Perturbation

For the decoding based on the original EDS or the modified EDS, we find that it may not converge or converge to non-optimal codewords. These problems influence the performances in the waterfall region and the error-floor region. To mitigate these problems, we propose two improved schemes based on the perturbation and the bit-flipping, respectively and introduce these two schemes in this section and the next section.

During the decoding, sometimes the sequential schedule may cause that a lot of check nodes are satisfied but each of them connects to even wrong variable nodes, as been shown in Fig. 4.1. In Fig. 4.1, unless the wrong variable node which connects to the unsatisfied check node is corrected, other wrong variable nodes can be corrected. Numerous variable nodes, which can not converge, in this subgraph result in errors. Rather than to avoid the occurrence of the problem, we try to identify those questionable variable nodes and let them converge. We assume that if the transmitted bits of those questionable variable nodes suffer different set of noise, they may converge and be decoded successfully.

With the modified EDS, every time before popping in new data, we compute average log-APP of variable nodes in each block in the window to decide the state of convergence.

If there are blocks not convergent after being decoded for a while, we suppose that these blocks will not be decode correctly. Then, we active perturbation algorithm to try to let variable nodes in these blocks converge. To speak elaborately, we add additional zero-mean Gaussian noise Δ_i with variance δ^2 to the received value y_i of a questionable variable nodes v_i. Then we reset the log-APPs of variable nodes in these blocks

$$Q_i = 2(y_i + \Delta_i)/\bar{\sigma}^2 = 2(x_i + (n_i + \Delta_i))/\bar{\sigma}^2 = 2\bar{y}_i/\bar{\sigma}^2,$$

where $\bar{\sigma}^2 = \sigma^2 + \delta^2$ and n_i is channel noise. After that, we execute hybrid operation PUC N_e times and check if these blocks are convergent or not. Here, δ^2 should be chosen carefully. It should not be neither too large nor too small. Since $\bar{y}_i$ can be viewed as that y_i suffered larger noise, if δ^2 is too large, decoding result may get worse. However, if δ^2 is too small, it may make no difference to the original result. Unfortunately, Δ_i's may fail. If it does, we generate another set of perturbation noise $\Delta_i^{'}$'s unless successive N_f tries fail. A block diagram of combination of perturbation algorithm and modified EDS is shown in Fig. 4.2.

Previously, perturbation algorithm (PA) is usually used to generate more candidate codewords. Besides, perturbation noise is added to the whole codeword. However, PA in this thesis is used to help variable nodes escape from the decoding trap, where parts of the Tanner graph can not converge. In addition, perturbation noise is added only to those questionable variable nodes.

4.3 Improved Scheme Based on Bit-Flipping

Several observations indicate that the decoding based on the original EDS or our improved EDS may converge to non-optimal codewords, which means that there is another codeword whose Euclidean distance is smaller than that between the decoded codeword and the received sequence, as shown in Fig. 4.3. To solve this problem, the idea is that we can generate several codeword candidates and choose the codeword with the best metric to be the decoding result, where the metric is the Euclidean distance between the codeword and the received sequence.

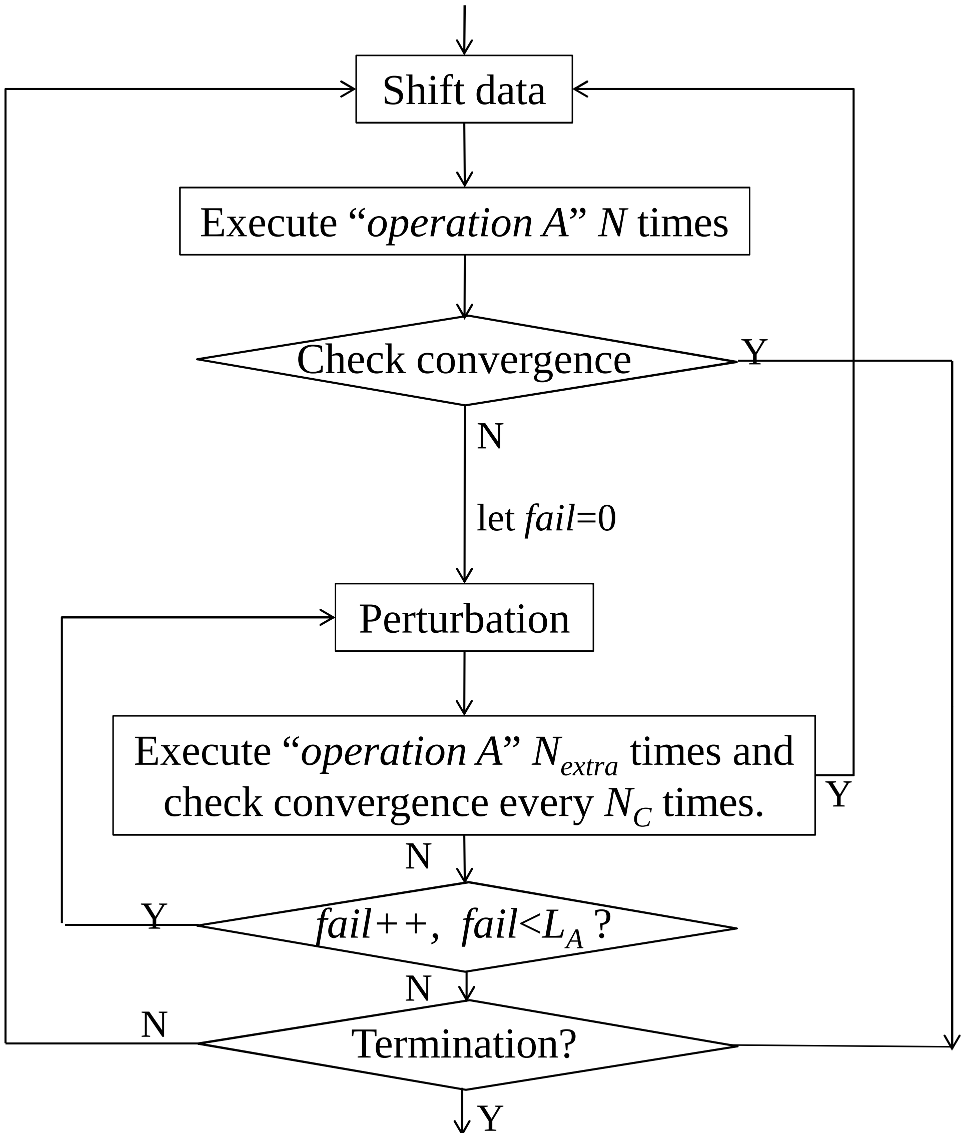

Shift data
Execute "operation A" N times
Check convergence
Y
N
let fail=0
Perturbation
Execute "operation A" N_extra times and check convergence every N_C times.
Y
N
fail++, fail<L_A ?
Y
N
Termination?
N
Y

Figure 4.2: Block diagram of combination of perturbation algorithm and modified EDS.

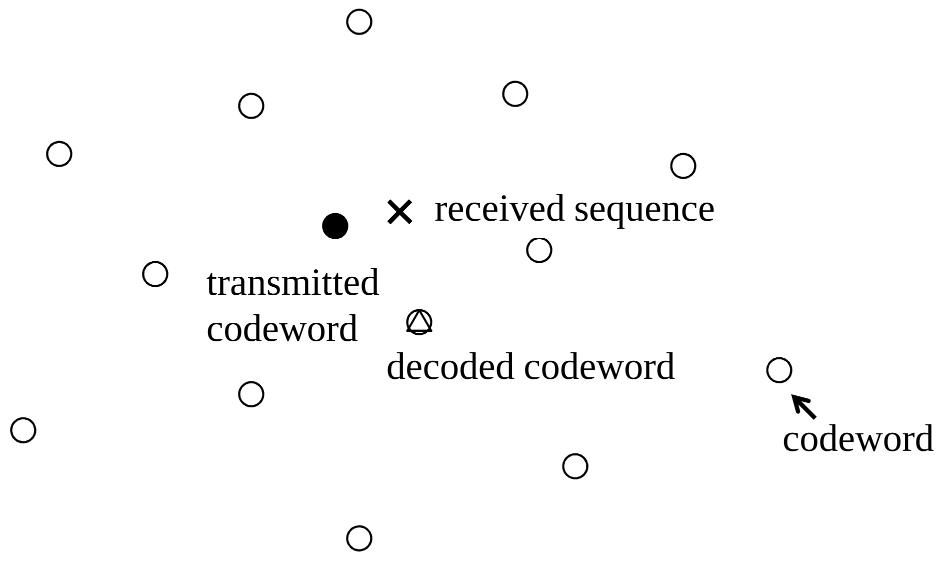


Figure 4.3: A sketch map of codeword space.

For LDPC-CCs, the difference between a codeword and one of its neighboring codewords is composed of one or several groups of variable nodes. Each group of variable nodes and all check nodes which connect to them can form a special subgraph. In this special subgraph, all check nodes connect to even variable nodes in the subgraph. For convenience, we call the relationship between variable nodes in a group pattern Γ_i. For an LDPC-CC, there can be infinite groups of variable nodes with the same pattern. Thus, $\Gamma_i(t_1)$ denotes the group of variable nodes whose variable nodes locate at time instant $\Gamma \geq \Gamma_1$. Through different patterns, we continuously check whether it is possible to get another codeword with a smaller Euclidean distance. However, assume that $\mathbf{y}$ is decoded into a wrong codeword with a larger Euclidean distance, and the difference between the decoded codeword and the transmitted codeword is composed of one group of variable nodes. Those wrong variable nodes in the group are usually caused by the decoding. Their received values may not be unreliable. Since to influence a group with many variable nodes is more difficult than a group with few ones, decoding errors are more likely to be variable nodes in small groups. Thus, only patterns of small groups are needed for improvement of the performance.

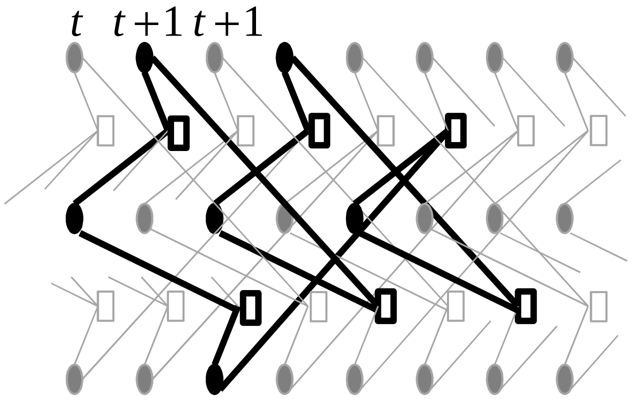


Figure 4.4: A special subgraph of the LDPC-CC in Ex. 3.1.

Example 4.1 *There's a group, which is composed of following variable nodes*

$$\begin{array}{ccccccc}
v_t^{(4)}, & v_{t+74}^{(4)}, & v_{t+84}^{(4)}, & v_{t+90}^{(5)}, & v_{t+106}^{(2)}, & v_{t+144}^{(1)}, & v_{t+154}^{(2)}, \\
v_{t+192}^{(2)}, & v_{t+194}^{(4)}, & v_{t+206}^{(4)}, & v_{t+212}^{(5)}, & v_{t+218}^{(1)}, & v_{t+228}^{(1)}, & v_{t+238}^{(2)}, \\
v_{t+250}^{(5)}, & v_{t+264}^{(1)}, & v_{t+268}^{(4)}, & v_{t+276}^{(2)}, & v_{t+278}^{(4)}, & v_{t+284}^{(5)}, & v_{t+290}^{(4)}, \\
v_{t+296}^{(5)}, & v_{t+300}^{(2)}, & v_{t+312}^{(1)}, & v_{t+348}^{(2)}, & v_{t+396}^{(1)}, & v_{t+406}^{(5)}, & v_{t+432}^{(2)},
\end{array}$$

in an LDPC-CC with $R = 0.5$ and $m_s = 203$,

$$\mathbf{H}(D) = \begin{bmatrix} 1+D^{194} & D^{158} & D^{166} & D^{144} & 0 & D^{65} & 0 & 0 \\ D^{97} & D^{49} & 0 & D^{203} & D^{65} & D^{37} & 1 & 0 \\ 0 & D^{106} & D^{83} & D^{138} & D^{48}+D^{132} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & D^{20} & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1+D^{76} & 1 \end{bmatrix}.$$

□

Assume that there is a group $\Gamma_i(t)$ going to leave the window. For $\Gamma_i(t)$, we firstly check its state of convergence and whether all related check nodes are satisfied. Since if these variable nodes are not convergent or some of related check nodes are not satisfied, the decoded sequence is not a codeword and there is no need to check whether there is a codeword with a smaller Euclidean distance. Then, we make hard decisions $\mathbf{z}(\Gamma_i(t))$ for variable nodes in $\Gamma_i(t)$. In addition, we compute the Euclidean distance $D_1(\Gamma_i(t))$ between $\mathbf{z}(\Gamma_i(t))$ and the corresponding received values $\mathbf{y}(\Gamma_i(t))$. Let $\bar{\mathbf{z}}(\Gamma_i(t)) = -\mathbf{z}(\Gamma_i(t))$. Afterward we compute

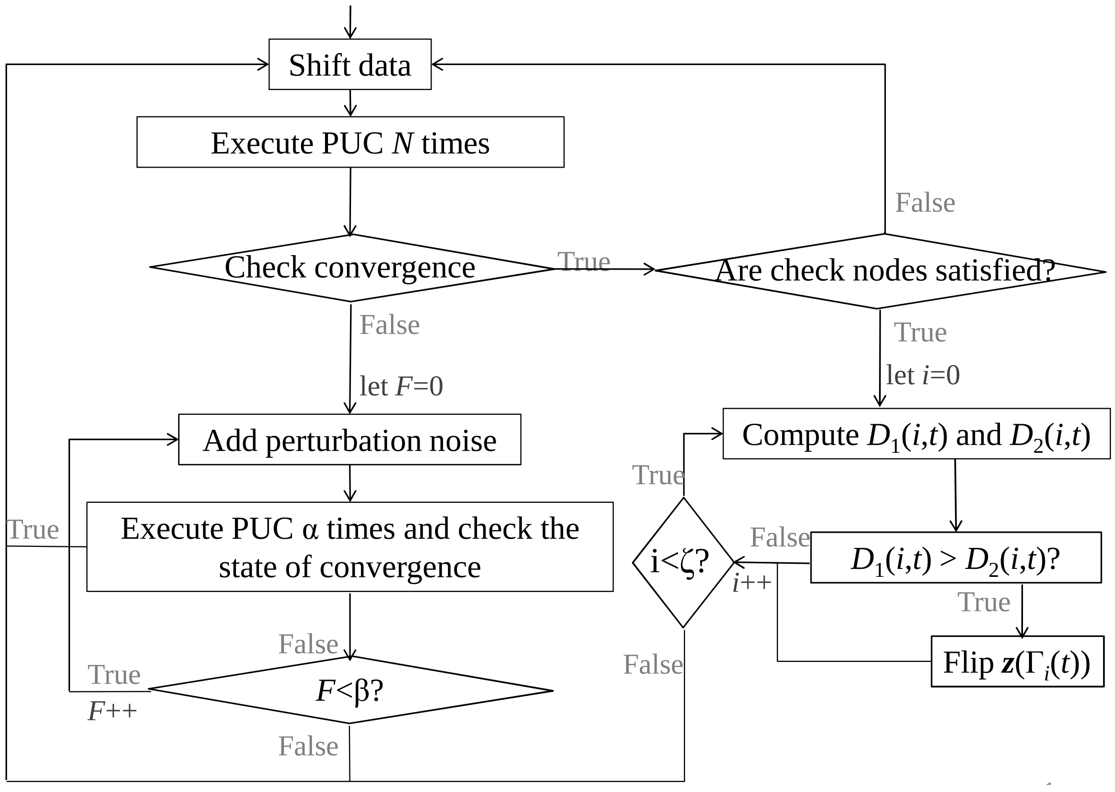


Figure 4.5: The block diagram of the proposed algorithm.

the Euclidean distance $D_2(\Gamma_i(t))$ between $\mathbf{z}'(\Gamma_i(t))$ and $\mathbf{y}(\Gamma_i(t))$. If $D_1(\Gamma_i(t)) > D_2(\Gamma_i(t))$, we flip all variable nodes in $\Gamma_i(t)$.

If the decoded codeword is farther from the received sequence than the correct codeword, as shown in Fig. 4.3, we may acquire the correct one by bit-flipping. However, if the decoded codeword is closer to the received sequence than the correct codeword, we can't rectify it. What's even worse, if the decoded codeword is the right one, but it is farther from the received sequence than others, we'll make a mistake. However, the probability of the occurrence of the first case is the highest. Thus, we can further improve the performance by this method.

Chapter 5

Simulation Results

In this chapter, we will show the performances of the proposed algorithm on a BPSK-modulated AWGN channel for LDPC-CCs with PPCMs and RPCMs.

Example 5.1 *Consider a rate $R = 0.5$ LDPC-CC with a PPCM*

$$\mathbf{H}_1(D) = \begin{bmatrix} 1+D^{194} & D^{158} & D^{166} & D^{144} & 0 & D^{65} \\ D^{97} & D^{49} & 1 & D^{203} & D^{65} & D^{37} \\ 0 & D^{106} & D^{83} & D^{138} & D^{48}+D^{132} & 1 \end{bmatrix}$$

with syndrome former memory $m_s = 203$. □

In Fig. 5.1(a), we show the performances of the flooding algorithm, the original EDS, our modified EDS, and the proposed algorithm. The number of iterations of the flooding algorithm is 100. For the original EDS, our modified EDS, and the proposed algorithm, let $N = 20400$ so that the amounts of check nodes being updated are the same with those of the flooding algorithm. Besides, K is 66 for the original EDS and our modified EDS. K is 50 for the proposed algorithm. Let $\alpha = 1000000$ and $\beta = 5$. Perturbation approximately results in an increase of 0.5 times of the computation complexity. As we can see, since the modified EDS has a more appropriate updating order, it has a better performance than that of the original EDS. In addition, with the aids of the perturbation and the bit-flipping, the proposed algorithm can further improve the performance. Comparing to the performance of the flooding algorithm, the performance of the proposed algorithm is better. In Fig. 5.1(b), we also show the performance of the flooding algorithm, whose number of iterations is 200.

200 is sufficient for the convergence of the flooding algorithm. Revealed by the simulation results, the performance of the proposed algorithm is also better than that of the flooding algorithm.

Example 5.2 *Consider a rate* $R = 0.4$ *LDPC-CC with a PPCM*

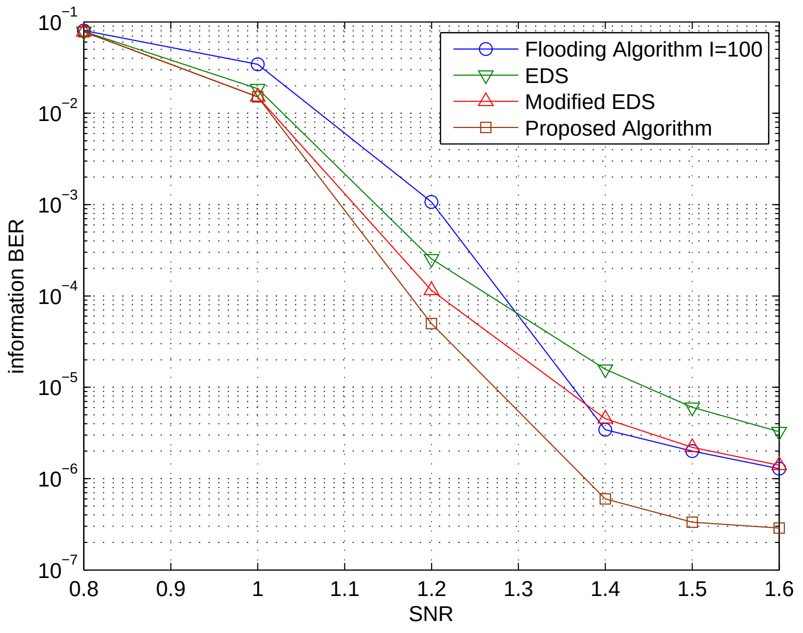
$$\mathbf{H}_2(D) = \begin{bmatrix} 1 & D^{251} & D^{353} & D^{376} & D^{278} \\ D^{32} & 1 & D^{356} & D^{395} & D^{119} \\ D^{256} & D^{37} & 1 & D^{359} & D^{312} \end{bmatrix}$$

with syndrome former memory $m_s = 395$. □

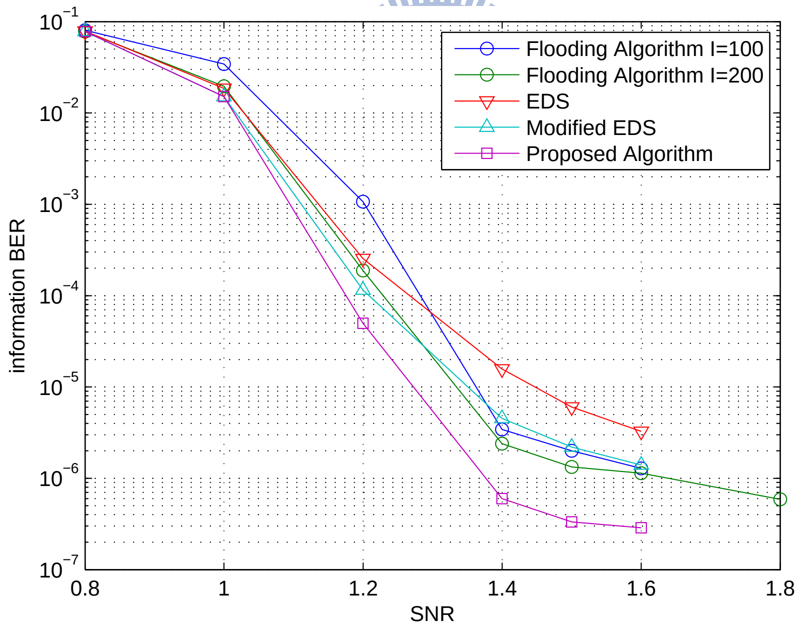
In Fig. 5.2(a), we show the BER performances of an $R = 0.4$ LDPC-CC with a PPCM. The number of iterations of the flooding algorithm is 100. For scheduling-based schemes, let $N = 39500$ so that the amounts of check nodes being updated are the same with those of the flooding algorithm. Besides, K is 66 for the original EDS and our modified EDS. K is 50 for the proposed algorithm. Let $\alpha = 1000000$ and $\beta = 5$. Perturbation also approximately results in an increase of 0.5 times of the computation complexity. As we can see, the performance of the modified EDS is better than that of the original EDS and the proposed algorithm further improve the performance. Comparing to the performance of the flooding algorithm, the performance of the proposed algorithm is better. In Fig. 5.2(b), the performance of the flooding algorithm, whose number of iterations is 500, is also shown. 500 is sufficient for the convergence of the flooding algorithm. Revealed by the simulation results, the performance of the proposed algorithm is also better than that of the flooding algorithm.

Example 5.3 *Consider* $\mathbf{H}_1(D)$ *in example* 1. *Now we replace its one polynomial term by a rational one and obtain*

$$\mathbf{H}_2'(D) = \begin{bmatrix} 1 + D^{194} & D^{158} & D^{166} & D^{144} & 0 & D^{65} \\ D^{97} & D^{49} & \frac{1}{1+D^{20}+D^{76}} & D^{203} & D^{65} & D^{37} \\ 0 & D^{106} & D^{83} & D^{138} & D^{48} + D^{132} & 1 \end{bmatrix}.$$

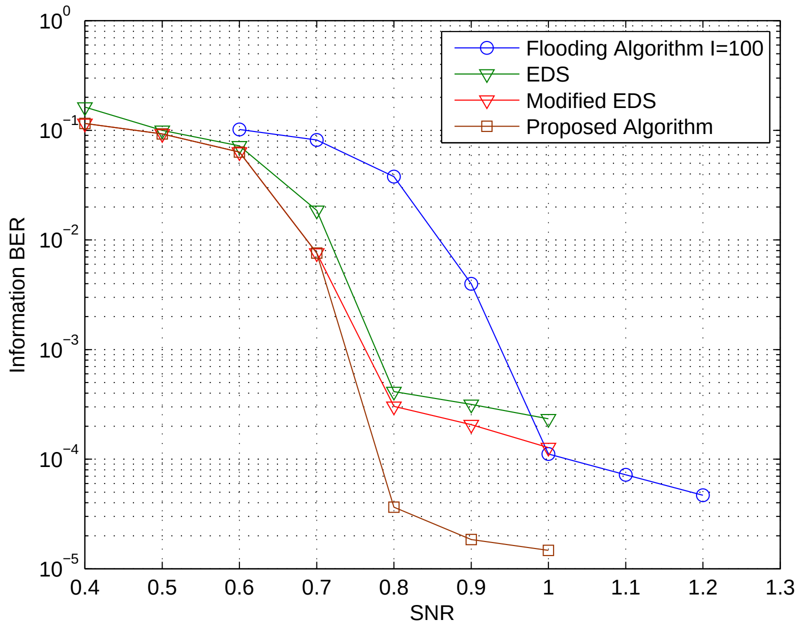


(a)

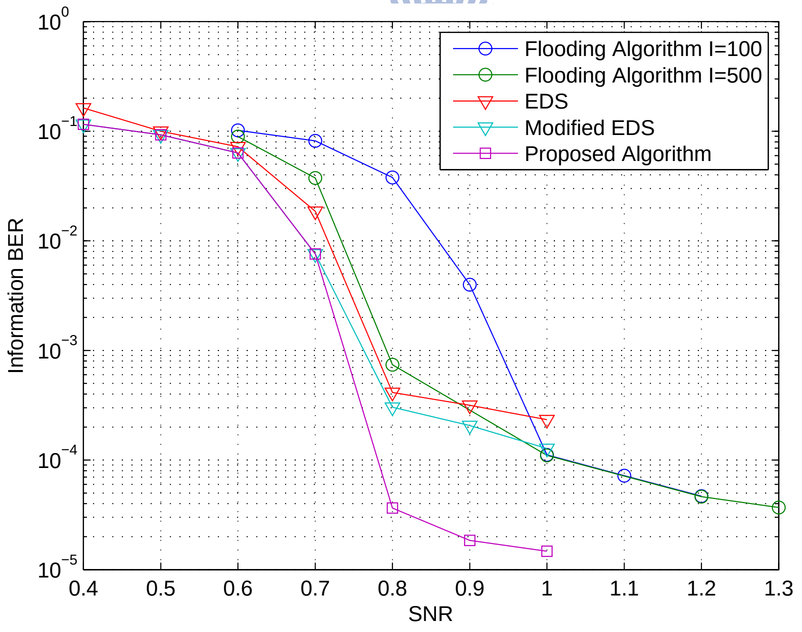


(b)

Figure 5.1: The BER performances of an $R = 0.5$ LDPC-CC with a PPCM, where $m_s = 203$.



(a)



(b)

Figure 5.2: The BER performances of an $R = 0.4$ LDPC-CC with a PPCM, where $m_s = 395$.

By using two variable nodes, $M_1(D)$ and $M_2(D)$, we expand it into a super code. Let

$$M_1(D) = \frac{1}{1+D^{20}+D^{76}} V_3(D),$$

rewrite the check equation

$$D^{97}V_1(D) + D^{49}V_2(D) + \frac{1}{1+D^{20}+D^{76}} V_3(D) + D^{203}V_4(D) + D^{65}V_5(D) + D^{37}V_6(D) = 0$$

into

$$D^{97}V_1(D) + D^{49}V_2(D) + D^{203}V_4(D) + D^{65}V_5(D) + D^{37}V_6(D) + M_1(D) = 0$$

, and additionally add a new check equation

$$V_3(D) + (1+D^{20}+D^{76})M_1(D) = 0.$$

Then, we obtain

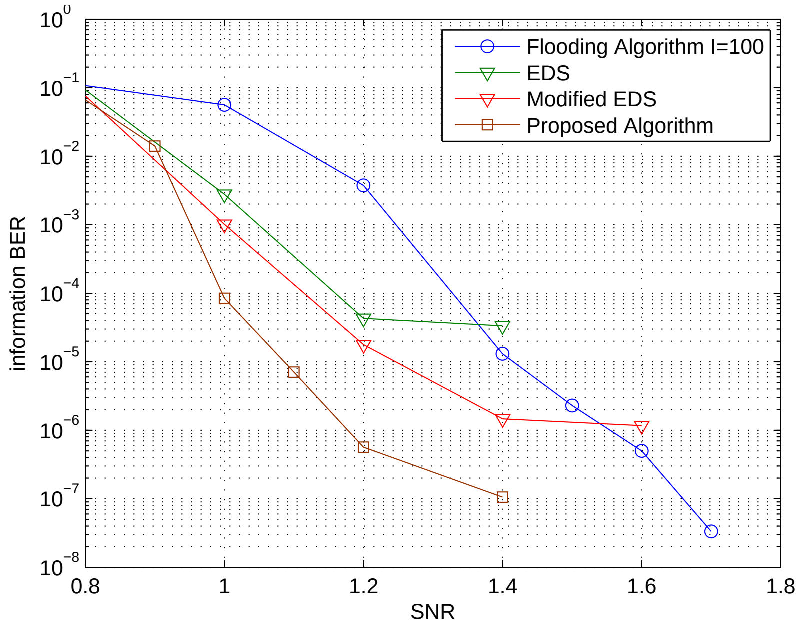
$$\mathbf{H}_2^{''}(D) = \begin{bmatrix} 1+D^{194} & D^{158} & D^{166} & D^{144} & 0 & D^{65} & 0 \\ D^{97} & D^{49} & 0 & D^{203} & D^{65} & D^{37} & 1 \\ 0 & D^{106} & D^{83} & D^{138} & D^{48}+D^{132} & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1+D^{20}+D^{76} \end{bmatrix}.$$

However, the trinomial term $1+D^{20}+D^{76}$ will results in small girth, thus we let $M_2(D) = (1+D^{76})M_1(D)$ and transform $\mathbf{H}_2^{''}(D)$ into

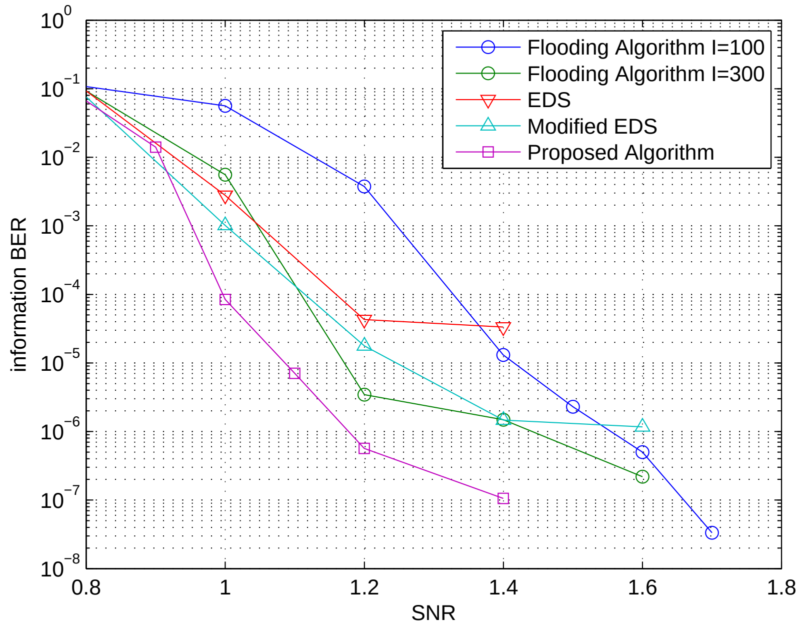
$$\mathbf{H}_2^{'''}(D) = \begin{bmatrix} 1+D^{194} & D^{158} & D^{166} & D^{144} & 0 & D^{65} & 0 & 0 \\ D^{97} & D^{49} & 0 & D^{203} & D^{65} & D^{37} & 1 & 0 \\ 0 & D^{106} & D^{83} & D^{138} & D^{48}+D^{132} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & D^{20} & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1+D^{76} & 1 \end{bmatrix}.$$

$\square$

To maintain the same code rate, we puncture variable nodes $V_3(D)$ and $M_1(D)$. In Fig. 5.3(a), we show the BER performances of an $R = 0.5$ LDPC-CC with an RPCM. The number of iterations of the flooding algorithm is 100. Let $N = 34000$ such that the amounts



(a)



(b)

Figure 5.3: The BER performances of an $R = 0.5$ LDPC-CC with an RPCM, where $m_s = 203$.

of check nodes being updated of other algorithms are the same with those of the flooding algorithm. Besides, K is 66 for the original EDS and our modified EDS. K is 50 for the proposed algorithm. Let $\alpha = 1000000$ and $\beta = 5$. Perturbation also approximately results in an increase of 0.5 times of the computation complexity. As we can see, the performance of the modified EDS is better than that of the original EDS and the proposed algorithm further improve the performance. Comparing to the performance of the flooding algorithm, the performance of the proposed algorithm is better. In Fig. 5.3(b), the performance of the flooding algorithm, whose number of iterations is 300, is also shown. 300 is sufficient for the convergence of the flooding algorithm. The performance of the proposed algorithm is also better than that of the flooding algorithm.

Chapter 6

Conclusion

In this thesis, we apply EDS to decode LDPC-CCs and modify the residual function of EDS to improve the BER performances. Besides, we analyze the decoded results of dynamic scheduling and propose two improved schemes based on the perturbation and the bit-flipping to further improve the BER performances. By the simulations results, the proposed algorithm is shown that it can provide a better BER performance comparing to that of the flooding algorithm for LDPC-CCs with rational parity-check matrices. In addition, for polynomial parity-check matrices, the proposed algorithm also performs well.

Bibliography

[1] A. Jiménez Felström and K. Zigangirov, "Time-varing periodical convolutional codes with low-density parity-check matrix," *IEEE Trans. Inf. Theory*, vol. 45, pp. 2181-2191, Sep. 1999.

[2] Guojun Han and Xingcheng Liu, "An efficient dynamic schedule for layered belief-propagation decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 13, pp. 950-952, Dec. 2009.

[3] Valentin Savin, "Iterative LDPC decoding using neighborhood reliabilities," in *Proc. IEEE Int. Symp. Inform. Theory*, Nice, France, June 2007, pp. 221-225.

[4] Hua Xiao and Amir H. Banihashemi, "Graph-based message-passing schedules for decoding LDPC codes," *IEEE Trans. on Communications*, vol. 52, pp. 2098-2105, Dec. 2004.

[5] Andres I. Vila Casado, Miguel Griot, and Richard D. Wesel, "Informed dynamic scheduling for belief-propagation decoding of LDPC codes," in *Proc. IEEE Int. Conf. Commun.*, Glasgow, Scotland, June 2007, pp. 932-937.

[6] Chih-Chieh Lai, "A study on LDPC-CC with rational parity-check metrices and related decoding algorithms," master thesis, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., 2009.

[7] Arvind Sridharan, "Design and analysis of LDPC convolutional dodes," Ph.D. dissertation, the University of Notre Dame, Indiana, U.S.A, 2005.

[8] J. Hahenauer, E. Offer, and L. Papke, “Iterative decoding of binary block and convolutional codes,” *IEEE Trans. Inf. Theory*, vol. 42, pp. 425-449, Mar. 1996.