Interleavers for Turbo Codes Using Permutation Polynomials Over Integer Rings

Jing Sun, *Student Member, IEEE,* and Oscar Y. Takeshita, *Member, IEEE*

***Abstract*—In this paper, a class of deterministic interleavers for turbo codes (TCs) based on permutation polynomials over $\mathbb{Z}_N$ is introduced. The main characteristic of this class of interleavers is that they can be algebraically designed to fit a given component code. Moreover, since the interleaver can be generated by a few simple computations, storage of the interleaver tables can be avoided. By using the permutation polynomial-based interleavers, the design of the interleavers reduces to the selection of the coefficients of the polynomials. It is observed that the performance of the TCs using these permutation polynomial-based interleavers is usually dominated by a subset of input weight $2m$ error events. The minimum distance and its multiplicity (or the first few spectrum lines) of this subset are used as design criterion to select good permutation polynomials. A simple method to enumerate these error events for small m is presented.**

Searches for good interleavers are performed. The decoding performance of these interleavers is close to S-random interleavers for long frame sizes. For short frame sizes, the new interleavers outperform S-random interleavers.

***Index Terms*—Integer ring, interleaver, permutation polynomial, turbo codes (TCs), weight spectrum.**

I. Introduction

A TYPICAL turbo code (TC) [1] is constructed by parallel concatenating two convolutional codes via an interleaver. The design of the interleaver is critical to the performance of the TC, particularly for short frame sizes. Interleavers for TCs have been extensively studied in [2]–[6]. They can in general be separated into two classes: random interleavers and deterministic interleavers.

The basic random interleavers permute the information bits in a pseudorandom manner. It was demonstrated in [1] that near-Shannon limit performance can be achieved with these interleavers for large frame sizes. The S-random interleaver proposed in [2] is an improvement to the random interleaver. The S-random interleaver is a pseudorandom interleaver with the restriction that any two input positions within distance S cannot be permuted to two output positions within distance S. Under this restriction, certain short error events in one component code will not be mapped to short error events in the other component code. Further improvements to the S-random interleaver have been presented in [3], [4]. In [3], an iterative decoding suitability (IDS) criterion is used to design two-step S-random interleavers. In [4], multiple error events are considered. These improved interleavers have a better performance than S-random interleavers, especially for short frame sizes.

TCs using random interleavers require an interleaver table to be stored both in the encoder and the decoder. This is not desirable in some applications, especially when the frame size is large or many different interleavers have to be stored [7]. Deterministic interleavers can avoid interleaver tables since interleaving and de-interleaving can be performed algorithmically.

The simplest deterministic interleavers are block interleavers [8] and linear interleavers [6]. For some component codes and very short frame sizes, block and linear interleavers, with properly chosen parameters, perform better than random interleavers. However, for medium to long frame sizes, they have a high error floor [6] and random interleavers are still better.

In [6], quadratic interleavers have been introduced. Quadratic interleavers are a class of deterministic interleavers based on a quadratic congruence. For $N = 2^n$, it can be shown that

$$c_i = \frac{ki(i+1)}{2} (\mathrm{mod} N)$$

is a permutation of $i \in \{0, 1, \ldots, N-1\}$ for odd k. Then the quadratic interleaver is defined as

$$\pi_{\mathcal{Q}_N} : c_i \mapsto c_{(i+1) \bmod N}, \quad \forall\, i$$

which means that the data in position c_i is interleaved to position $c_{(i+1) \bmod N}$, for all i. The quadratic interleavers perform well. Even though they are not as good as S-random interleavers, it is claimed in [6] that they achieve the average performance of random interleavers. Due to this characteristic, in the simulations in Section V, in addition to S-random interleavers, we will compare our proposed interleavers to quadratic interleavers instead of the average of many randomly generated interleavers.

Recently, some other deterministic interleavers have been proposed. In [9], dithered relative prime (DRP) interleavers are used. A good minimum distance property can be achieved for short frame size cases by carefully selecting the parameters of the interleaving process. In [10], monomial interleavers are used, which are based on a special permutation polynomial in the form of x^i over a finite field. This class of interleavers can also achieve a similar performance as random interleavers. However, since the permutation polynomials are over a finite field, the frame size can only be a power of a prime.

Manuscript received August 20, 2003; revised July 21, 2004. This work was supported by the National Science Foundation under Grants CCR-00752219 and ANI-9809018.

J. Sun was with the Electrical and Computer Engineering Department, the Ohio State University, Columbus OH 43210 USA. He is now with Qualcomm, San Diego, CA 92121 USA (e-mail: sunj@ieee.org).

O. Y. Takeshita are with the Electrical and Computer Engineering Department, the Ohio State University, Columbus OH 43210 USA (e-mail: Takeshita.3@osu.edu).

Communicated by Ø. Ytrehus, Associate Editor for Coding Techniques.
Digital Object Identifier 10.1109/TIT.2004.839478

In this paper, we will propose another class of deterministic interleavers. These interleavers are based on permutation polynomials over the ring of integers modulo N, $\mathbb{Z}_N$. By carefully selecting the coefficients of the polynomial, we can achieve a performance close to, or in some cases, even better than S-random interleavers.

It has been observed that a subset of error events with input weight $2m$, m being a small positive integer, usually dominates the performance, at least when the frame size is not very short. The parameter selection of the permutation polynomial is based on the minimum distance or the first few spectrum lines of this subset. For small m, we present a simple method to find these error events. This helps to reduce the complexity of searching for good permutation polynomial-based interleavers given the component code.

The paper is organized as follows. In Section II, we briefly review permutation polynomials over $\mathbb{Z}_N$. Interleavers based on permutation polynomials are introduced in Section III. The minimum distance and the spectrum based analysis are given in Section IV. Simulation results are shown in Section V, and finally, we draw our conclusions in Section VI.

II. Permutation Polynomials Over $\mathbb{Z}_n$

Given an integer $N \geq 2$, a polynomial

$$P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_mx^m$$

where $a_0, a_1, \ldots, a_m$ and m are nonnegative integers, is said to be a permutation polynomial over $\mathbb{Z}_N$ when $P(x)$ permutes $\{0, 1, 2, \ldots, N-1\}$. In this paper, all the summations and multiplications are modulo N unless explicitly stated. We further define the *formal derivative* of the polynomial $P(x)$ to be a polynomial $P'(x)$ such that

$$P'(x) = a_1 + 2a_2x + 3a_3x^2 + \cdots + ma_mx^{m-1}. \quad (1)$$

For the special case that $N = 2^n$, the necessary and sufficient condition for a polynomial to be a permutation polynomial is shown in [11]. It is repeated in the following theorem.

Theorem 2.1: Let $P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_mx^m$ be a polynomial with integer coefficients. $P(x)$ is a permutation polynomial over the integer ring $\mathbb{Z}_{2^n}$ if and only if 1) a_1 is odd, 2) $a_2 + a_4 + a_6 + \cdots$ is even, and 3) $a_3 + a_5 + a_7 + \cdots$ is even.

For the more general case $N = p^n$, where p is any prime number, the necessary and sufficient condition is derived in [12].

Theorem 2.2: $p(x)$ is a permutation polynomial over the integer ring $\mathbb{Z}_{p^n}$ if and only if $P(x)$ is a permutation polynomial over $\mathbb{Z}(p)$ and $P'(x) \neq 0$ modulo p for all integers $x \in \mathbb{Z}_{p^n}$.

For example, let $P(x) = 3x^2 + x$ and $p = 3$. Then $P(0) = 0 \bmod 3$, $P(1) = 1 \bmod 3$, and $P(2) = 2 \bmod 3$, which implies that $P(x)$ is a permutation polynomial over $\mathbb{Z}(p)$. The formal derivative of $P(x)$ is $P'(x) = 6x+1 = 1 \bmod 3$, which is a nonzero constant for all x in $\mathbb{Z}_{p^n}$. Thus, Theorem 2.2 is satisfied and $P(x)$ is a permutation polynomial over $\mathbb{Z}_{p^n}$.

It can be easily verified that when $p = 2$, the necessary and sufficient condition in Theorem 2.2 reduces to the form in Theorem 2.1. However, since in many of our design examples we use $N = 2^n$, we keep Theorem 2.1 for its simplicity.

For general N, we also have a necessary and sufficient condition for a polynomial to be a permutation polynomial, which is summarized in the following theorem.

Theorem 2.3: For any $N = \prod_{i=1}^{m} p_i^{n_{p_i}}$, where p_i's are distinct prime numbers, $P(x)$ is a permutation polynomial modulo N if and only if $P(x)$ is also a permutation polynomial modulo $p_i^{n_{p_i}}, \forall\, i$.

Proof: First we prove the necessity.

Let $P(x)$ be a permutation polynomial over $\mathbb{Z}_N$. For any l

$$\begin{aligned} P\left(x + lp_i^{n_{p_i}}\right) &= a_0 + a_1\left(x + lp_i^{n_{p_i}}\right) + a_2\left(x + lp_i^{n_{p_i}}\right)^2 \\ &\quad + \cdots + a_m\left(x + lp_i^{n_{p_i}}\right)^m \\ &= a_0 + a_1x + a_2x^2 + \cdots \\ &\quad + a_mx^m \bmod p_i^{n_{p_i}}. \end{aligned}$$

So $P\left(x + lp_i^{n_{p_i}}\right) = P(x) \bmod p_i^{n_{p_i}}$.

Assume that $P(x)$ is not a permutation polynomial over $p_i^{n_{p_i}}$. Then there exist $x_1 \neq x_2$, $0 \leq x_1, x_2 < p_i^{n_{p_i}}$ such that

$$P(x_1) = P(x_2) = y \bmod p_i^{n_{p_i}}.$$

Then $\forall\, l \in \{0, 1, \ldots, \frac{N}{p_i^{n_{p_i}}} - 1\}$

$$P\left(x_1 + lp_i^{n_{p_i}}\right) = P\left(x_2 + lp_i^{n_{p_i}}\right) = y \bmod p_i^{n_{p_i}}.$$

So a total of $\frac{2N}{p_i^{n_{p_i}}}$ numbers are mapped to $y \bmod p_i^{n_{p_i}}$. This cannot be true since $P(x)$ is a permutation polynomial modulo N and it can map exactly $\frac{N}{p_i^{n_{p_i}}}$ numbers to $y \bmod p_i^{n_{p_i}}$.

Next, we prove the sufficiency. Let N_1 and N_2 be coprime. Let $P(x)$ be a permutation polynomial both modulo N_1 and N_2. Assume that $P(x)$ is not a permutation polynomial modulo N_1N_2. Then there exist x and x' such that $x \neq x' \bmod N_1N_2$ and $P(x) = P(x') \bmod N_1N_2$. Define $x_1 = x \bmod N_1$, $x_2 = x \bmod N_2$ and $x'_1 = x' \bmod N_1$, $x'_2 = x' \bmod N_2$. Since $P(x) = P(x') \bmod N_1N_2$, we have $P(x_1) = P(x'_1) \bmod N_1$ and $P(x_2) = P(x'_2) \bmod N_2$. Since $P(x)$ is a permutation polynomial modulo N_1 and N_2, we must have $x_1 = x'_1 \bmod N_1$ and $x_2 = x'_2 \bmod N_2$. By the Chinese Remainder Theorem [13], this implies that $x = x' \bmod N_1N_2$, which contradicts the assumption. Thus, $P(x)$ is also a permutation polynomial modulo N_1N_2. □

Using this theorem, checking whether a polynomial is a permutation polynomial modulo N reduces to checking the polynomial modulo each $p_i^{n_{p_i}}$ factor of N.

For $p_i = 2$, we can use Theorem 2.1 to check if the polynomial is a permutation polynomial, which is a simple test on the coefficients. For general p_i, we must use Theorem 2.2, which cannot be done by simply looking at the coefficients. In the following, we will derive some simple check criteria for some subclasses of permutation polynomials.

If we limit the polynomials to be of second degree, we have the following corollary.

Corollary 2.4: A second degree polynomial of the form $P(x) = ax + bx^2$ is a permutation polynomial over $\mathbb{Z}_{p^n}$ if and only if $a \neq 0$ and $b = 0 \bmod p$.

Proof: When $p = 2$, this corollary is a special case of Theorem 2.1. So we only need to consider the case that $p \neq 2$. In order to prove the corollary, we only need to check the conditions in Theorem 2.2.

Let $P(x)$ be a permutation polynomial over $\mathbb{Z}_{p^n}$, then from the second condition in Theorem 2.2, the formal derivative $a + 2bx \neq 0$ modulo p for all $x \in [0, \ldots, p-1]$. Let $x = 0$, the second condition implies that $a \neq 0 \bmod p$. Assume $b \neq 0 \bmod p$, then $2b$ is coprime to p and $2bx$ permutes $[0, \ldots, p-1]$. This implies that there exists a certain x such that $a + 2bx = 0$, which contradicts the condition. Thus, $b = 0 \bmod p$.

On the other hand, if both $a \neq 0 \bmod p$ and $b = 0 \bmod p$ are satisfied, then $P(x)$ can be reduced to $P(x) = ax$ modulo $\mathbb{Z}(p)$, which is a permutation polynomial. The formal derivative $P'(x) = a \neq 0$. The conditions for Theorem 2.2 are satisfied and $P(x)$ is a permutation polynomial over $\mathbb{Z}_{p^n}$. $\square$

Another class of permutation polynomials over $\mathbb{Z}_{p^n}$ can be "generated" using permutation polynomial $P_{\mathrm{GF}}(x) = ax$ over $\mathbb{Z}(p)$, where p is a prime number. This is summarized in the following corollary.

Corollary 2.5: If $P_{\mathrm{GF}}(x) = ax$ is a permutation polynomial over $\mathbb{Z}(p)$ for prime p, define $P_P(x) = \sum b_i x^i$, where $b_i = 0 \bmod p$, then $P_R(x) = P_{\mathrm{GF}}(x) + P_P(x)$ is a permutation polynomial over $\mathbb{Z}_{p^n}$.

Proof: Since $P_{\mathrm{GF}}(x) = ax$ is a permutation polynomial over $\mathbb{Z}(p)$ for prime p, we must have $a \neq 0 \bmod p$. Since $P_P(x) = 0 \bmod p$, then $P_R(x) = P_{\mathrm{GF}}(x) \bmod p$ and $P_R(x)$ is a permutation polynomial over $\mathbb{Z}(p)$. The formal derivative

$$P'_P(x) = \sum i b_i x^{i-1} = 0 \bmod p.$$

Then $P'_R(x) = P'_P(x) + a = a \neq 0 \bmod p$. Both of the two conditions in Theorem 2.2 are satisfied. $\square$

To simplify the notations in the paper, we need to introduce some definitions.

Definition 2.6 (Base): For $N = \prod_{i=1}^m p_i^{n_{p_i}}$, we define the *base* of N to be the vector $p_N = [p_1, p_2, \ldots, p_m]$.

Definition 2.7 (Order of N): For $N = \prod_{i=1}^m p_i^{n_{p_i}}$, we define the *order* of N to be the vector $o_N = [n_{p_1}, n_{p_2}, \ldots, n_{p_m}]$.

With respect to N, for any given number x, we can write it in the form $x = x_0 \prod_{i=1}^m p_i^{o_x[i]}$, where the p_i's are the elements in p_N and x_0 is coprime to p_i for any i.

Definition 2.8 (Order of x With Respect to N): We can define the *order* of x (with respect to N) by $o_x = [o_x[1], o_x[2], \ldots, o_x[m]]$.

The elements of any order vectors must be nonnegative. We also define $p_N^{o_x}$ to be $\prod_{i=1}^m p_N[i]^{o_x[i]}$. For example, if $N = 720 = 2^4 \cdot 3^2 \cdot 5^1$, then $p_N = [2,3,5]$ and $o_N = [4,2,1]$. If $x = 12 = 2^2 \cdot 3^1 \cdot 5^0$, then $o_x = [2,1,0]$ and $p_N^{o_x} = 2^2 \cdot 3^1 \cdot 5^0 = 12$. If $y = 35 = 2^0 \cdot 3^0 \cdot 5^1 \cdot 7^1$, then $o_y = [0,0,1]$ and $p_N^{o_y} = 2^0 \cdot 3^0 \cdot 5^1 = 5$. In this paper, by default, the orders of all numbers are with respect to N.

Since our computations are modulo N, we need to define a special comparison of orders. First, we define the comparison of orders in each dimension. Given N with base p_N and order o_N, for two numbers x and y, their orders are o_x and o_y, respectively. Basically, the comparison for the ith elements in o_x and o_y is based on the result of comparison of x and y modulo $p_i^{o_N[i]}$. More precisely, we define $o_x[i] = o_y[i]$ if both of them are not less than $o_N[i]$. Otherwise, define the ordering of $o_x[i]$ and $o_y[i]$ by their numerical ordering.

Based on the comparisons of orders in each dimension, we can define comparisons of orders. We define $o_x \le o_y$ if $o_x[i] \le o_y[i]$ for all i. We define $o_x < o_y$ if $o_x \le o_y$ and for at least one i, $o_x[i] < o_y[i]$. Other comparisons like $o_x > o_y$, $o_x \ge o_y$, and $o_x = o_y$ are similarly defined. Note that by the above definition, the order is a partial ordered set. $o_x \not< o_y$ does not imply $o_x \ge o_y$. We further define o_x strictly less than o_y, denoted by $o_x \ll o_y$, to be the case that $o_x[i] < o_y[i]$ for all i. Strictly greater than is similarly defined.

We also define some operations on orders. The summation and subtraction of orders are defined to be element-wise. However, since orders must be nonnegative, subtraction $o_z = o_x - o_y$ is only defined when $o_x \ge o_y$. Other operations like $\max(\cdot,\cdot)$ and $\min(\cdot,\cdot)$ are also defined to be element-wise.

Orders of numbers have some useful properties.

Property 2.9: If $z = xy$, then $o_z = o_x + o_y$.

Property 2.10: If x divides y, let $z = \frac{y}{x}$, then $o_z = o_y - o_x$.

Property 2.11: Let $z = x^y$, then $o_z = y o_x$.

The proofs for these properties are simple. Furthermore, when $o_x \gg o_y$, if $z = x + y$, then $o_z = o_y$. This result is not as trivial as the previous ones and we will give a simple proof. Let $x = x_0 \prod_{i=1}^m p_i^{o_x[i]}$ and $y = y_0 \prod_{i=1}^m p_i^{o_y[i]}$. By the definition of order, x_0 and y_0 are coprime to all p_i's. Since $o_x \gg o_y$

$$\begin{aligned} z &= x + y \\ &= x_0 \prod_{i=1}^m p_i^{o_x[i]} + y_0 \prod_{i=1}^m p_i^{o_y[i]} \\ &= \prod_{i=1}^m p_i^{o_y[i]} \left(x_0 \prod_{i=1}^m p_i^{o_x[i] - o_y[i]} + y_0 \right) \end{aligned} \tag{2}$$

and $o_x[i] - o_y[i]$ is positive for all i. Since y_0 is coprime to all p_i's, the term

$$\left(x_0 \prod_{i=1}^m p_i^{o_x[i] - o_y[i]} + y_0 \right)$$

is also coprime to all p_i's. Thus, we can observe from (2) that $o_z = o_y$.

III. Permutation Polynomial-Based Interleavers

An interleaver needs to permute the numbers in $\{0, 1, 2, \ldots, N-1\}$. This is exactly what a permutation polynomial can do. For example, when $N = 8$, polynomial $P(x) = 2x^2 + x + 3$ satisfies the conditions in Theorem 2.1 and is a permutation polynomial over $\mathbb{Z}_N$. If we interleave x to $P(x)$, then the sequence $\{0,1,2,3,4,5,6,7\}$ will be interleaved to $\{3,6,5,0,7,2,1,4\}$.

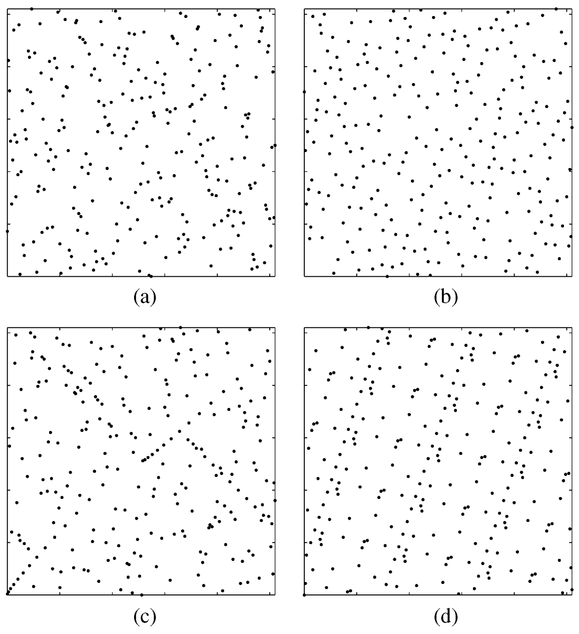

Fig. 1. Different interleavers of length 256. (a) Random, (b) S-random, (c) quadratic, (d) permutation polynomial based, $P(x) = 8x^2 + 3x$.

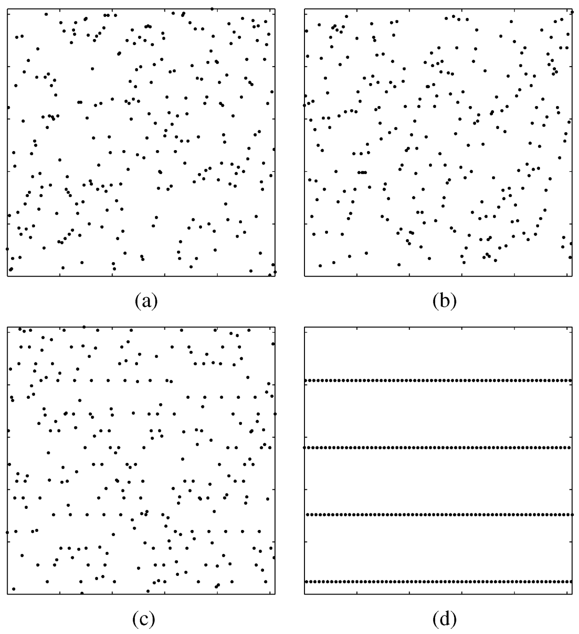

Fig. 2. $\Delta(x, 3)$ for different interleavers of length 256. (a) Random, (b) S-random, (c) quadratic, (d) permutation polynomial based, $P(x) = 8x^2 + 3x$.

In general, if a polynomial $P(x)$ is a permutation polynomial over $\mathbb{Z}_N$, then an interleaver based on this permutation polynomial can be defined as

$$\pi_{\mathcal{P}_N} : x \mapsto P(x), \quad \forall\, x.$$

In Fig. 1, plots of some interleavers are shown. In these plots, the x-axis is the input bit position and the y-axis is the output bit position. Comparing random and S-random interleavers (Fig. 1(a) and (b)), it can be observed that the points in S-random interleaver are distributed more uniformly in the plane. This property can help to avoid short error events in one component code to be interleaved to short error events in the other component code. Fig. 1(c) is for the quadratic interleaver. It resembles the random interleaver in the way that we can also observe irregularity in the density of points in the plane. Fig. 1(d) corresponds to an interleaver generated by the permutation polynomial $P(x) = 8x^2 + 3x$. The density of points in the plane is also not uniform. This characteristic is similar to random and quadratic interleavers. We can also observe that there are some periodic patterns in the plot. This regularity does not necessarily imply a bad performance, which can be better explained by considering the input weight 2 error events.

An input weight 2 error event is defined as an error event with two information bits in error. Each input weight 2 error event of a TC corresponds to one input weight 2 error event in each of the two component codes. We know that in TCs using random interleavers, input weight 2 error events decide the effective free distance and further decide the performance of the code in the error floor region (high signal-to-noise ratio (SNR) [14]. For permutation polynomial-based interleavers, this type of error events is also an important factor. An input weight 2 error event in the first component code can be represented by $(x, x + t)$. (The pair indicates the locations of the two 1's in the error event.) These two positions will be interleaved to $\pi(x)$ and $\pi(x + t)$ in the second component code. The distance between $\pi(x)$ and $\pi(x + t)$ is denoted by

$$\Delta(x, t) = \pi(x + t) - \pi(x) \bmod N.$$

If we fix t, we can plot $\Delta(x, t)$ with respect to x. We are interested in points where $\pi(x + t)$ and $\pi(x)$ are close to each other. When we talk about points near $\Delta(x, t) = 0$, we mean points close to the line from both above and below (near the top of the plot).

For $t = 3$, the distance $\Delta(x, t)$'s for interleavers in Fig. 1 are shown in Fig. 2. Fig. 2(a)–(d) is for random, S-random, quadratic, and permutation polynomial-based interleavers, respectively. The plots for random and quadratic interleaver are very similar and look random. At first glance, the plot for the S-random interleaver is also similar to Fig. 2(a) and (c). But after a closer look, one can notice that there are no points near the $\Delta(x, t) = 0$ line. This is because for this S-random interleaver with $S = 8$, since $t \leq S$, $S < \Delta(x, t) < N - S$ for all x. For a permutation polynomial-based interleaver, the $\Delta(x, t)$ is very regular. All the points are uniformly located along a few equally separated horizontal lines. Similar to the S-random interleaver, if the coefficients in $P(x)$ are carefully chosen, there are no points near the $\Delta(x, t) = 0$ line. The remaining problem is how to select good coefficients. This is discussed in Section IV.

IV. Permutation Polynomials Search

Interleavers based on different permutation polynomials have different performances. We are interested in finding the best permutation polynomial for a given component code and a given frame size. For a fixed frame size N, the only variables are the degrees and the coefficients of the permutation polynomials.

In this paper, we will focus on second-degree polynomials $P(x) = bx^2 + ax + c$. The first reason for this selection is to have the lowest possible complexity. A first-degree polynomial $P(x) = ax + c$ (linear interleavers) is the simplest one among all permutation polynomials. However, as shown in [6], linear interleavers have very bad input weight 4 error event characteristics, causing a high error floor at medium to long frame size cases. This leaves the second-degree polynomial the next in the chain. Another reason to consider second-degree polynomials is their relatively easy analysis. This will be shown in Sections IV-A–C.

Notice that the constant term c just corresponds to a cyclic rotation in the interleaved sequence. It does not appear in the conditions for $P(x)$ to be a permutation polynomial and it does not affect the performance of the concatenated system if we disregard boundary effects. Therefore, we can simply let it be zero and consider polynomials of the form $P(x) = bx^2 + ax$.

In this section, we will use the minimum distance (or the first few spectrum lines) of a subset of error events of the TC as a criterion to select the coefficients of the polynomials. Namely, we will only consider some error events with an input weight $2m$. Although these error events do not cover all possible error events, due to the structure of the permutation polynomial-based interleavers, these error events have high multiplicity and the performance of the TC is usually dominated by them, especially when the frame size is not very short. The benefit of limiting to this subset is that these error events are relatively easy to be found and counted. The minimum distance for the limited set is usually close to the real minimum distance and can be used as an upper bound. We can also find the true minimum distance or the first few spectrum lines of the TC using the method in [15]. However, the complexity of doing this is relatively high.

In this paper, we assume that for each of the component convolutional codes, a tail-biting trellis [16] is used. The end of the trellis is directly connected to the start of the trellis and no flushing bits are used. For tail-biting trellises, a cyclic shift of an error event in one component code is also an error event. Furthermore, it is possible that an error event starts near the end of the trellis and wraps over to the head of the trellis. Due to this phenomenon, we say that the error events are modulo N.

The tail-biting assumption is used to simplify our analysis. By using this assumption, we can ignore any boundary effects due to termination, which helps in finding and counting the error events we are interested in. Unfortunately, since we are using systematic recursive convolutional codes as component codes, a tail-biting trellis does not always exist [17]. Some form of termination has to be used. Some modulo N error events will be broken by termination and the termination itself may introduce some extra error events. If we still find and count the error events modulo N, some error will be introduced. However, the proportion of the modulo N error events broken by termination is very small and the low-weight error events introduced by the termination are usually of low multiplicities. Therefore, when the frame size is not very short, the modulo N error events dominate the performance. We can still count the error events modulo N and ignore the error introduced in the weight spectrum estimation.

A. Input Weight $2m$ Error Events

Long random interleavers can be approximated by *uniform interleavers*, which is a probabilistic device that interleaves a given input position to any possible output positions with an

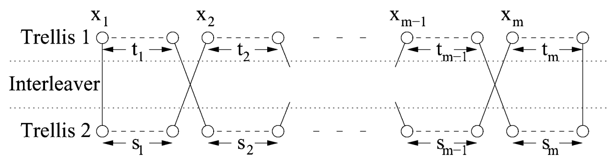

Fig. 3. Input weight $2m$ error events.

equal probability [14]. Using the uniform interleaver model, for very high-SNR region, the decoding performance has been shown to be dominated by input weight 2 error events. The minimum distance associated with input weight 2 error events is called effective free distance d_{ef} [2] and has been used as a design criterion to select component codes to have a good error floor performance.

Permutation polynomial-based interleavers are not random. In some sense, they cannot be well approximated by a uniform interleaver. So considering the effective free distance may not be enough. For example, for a linear permutation polynomial $P(x) = ax$, by carefully selecting the parameter a, the effective free distance can be designed to be very large. However, as analyzed in [6], for a certain type of input weight 4 error events, the corresponding Hamming distance is small. Furthermore, the number of these input weight 4 error events is proportional to the length of the frame, and cannot be controlled by selecting a. At least for a medium to high channel SNR, the performance is dominated by these input weight 4 error events.

We can think of the permutation polynomial-based interleavers as generalizations to the linear interleavers. It is observed that the most frequent error events are of input weight $2m$ where $m = 1, 2, \ldots$, just like in the linear case. However, the number of these error events can be controlled by selecting the parameters a and b. Given the component codes, we can find parameters pairs with a good performance.

A typical input weight $2m$ error event is shown in Fig. 3. This error event is composed of m input weight 2 error events (represented by dashed line segments) in each component code and these error events are interconnected via the interleaver (solid lines). In the figure, the ith input weight 2 error event in the first component code begins at x_i and has length $t_i + 1$. The ith input weight 2 error event in the second component code has length $s_i + 1$. Each input weight 2 error event can be represented by a pair of integers indicating the starting and ending positions in the corresponding trellis. Since we use the same systematic recursive convolutional code for both component codes, all t_i's and s_i's are multiples of the *cycle length* τ of the convolutional code. Here, the cycle length τ is defined as the cycle of the output of the encoder when the input sequence is $[1, 0, 0, 0, \ldots]$. For example, for the recursive convolutional code with generating polynomial $\frac{1+D^2}{1+D+D^2}$ (from now on, we will use the simple octal notation, $5/7$ for this example), with the previously mentioned input, the output sequence is $[1, 1, 1, 0, 1, 1, 0, 1, 1, 0, \ldots]$. There is a cycle of $[1, 1, 0]$ in the output and the cycle length $\tau = 3$. The cycle length equals the length of the shortest input weight 2 error event minus 1. We define *error pattern* to be the length $2m$ vector $[t_1, t_2, \ldots, t_m, s_1, s_2, \ldots, s_m]$.

For this input weight $2m$ error event, we have

$$P(x_2) - P(x_1) = s_1 \quad (3.1)$$
$$P(x_3) - P(x_1 + t_1) = s_2 \quad (3.2)$$
$$P(x_4) - P(x_2 + t_2) = s_3 \quad (3.3)$$
$$\vdots$$
$$P(x_{m-1}) - P(x_{m-3} + t_{m-3}) = s_{m-2} \quad (3.m-2)$$
$$P(x_m) - P(x_{m-2} + t_{m-2}) = s_{m-1} \quad (3.m-1)$$
$$P(x_m + t_m) - P(x_{m-1} + t_{m-1}) = s_m. \quad (3.m)$$

For such an input weight $2m$ error event to be formed, all these m equations need to be satisfied. There are a total of $3m$ variables in these m equations. Among these $3m$ variables, the x_i's take value from 0 to $N-1$ and both t_i's and s_i's are multiples of the cycle length τ. We will only consider t_i's and s_i's to be small multiples of τ since large values are associated with large Hamming distances.

To simplify the process of finding the error event, we will convert (3) to a form that is easier to analyze. This is done by computing $s_1 - s_2 + s_3 - s_4 + \cdots$. Using (3) and after some manipulations, we have

$$\begin{aligned} s_1 - s_2 + s_3 - s_4 \cdots = {} & 2b(x_1t_1 - x_2t_2 + x_3t_3 - x_4t_4 \cdots) \\ & + b(t_1^2 - t_2^2 + t_3^2 - t_4^2 \cdots) \\ & + a(t_1 - t_2 + t_3 - t_4 \cdots) \end{aligned} \quad (4)$$

or simply

$$\begin{aligned} \sum_{i=1}^{m} (-1)^{i-1} s_i = {} & 2b \sum_{i=1}^{m} (-1)^{i-1} x_i t_i \\ & + b \sum_{i=1}^{m} (-1)^{i-1} t_i^2 + a \sum_{i=1}^{m} (-1)^{i-1} t_i. \end{aligned} \quad (5)$$

Of course, (5) by itself is not a necessary and sufficient condition for such an input weight $2m$ error event to appear. It needs to be combined with any $m-1$ out of m equations in (3). This set of equations will be used in the rest of the paper to check for an input weight $2m$ error event.

Given an error pattern, if the $2m$ first-order error events do not overlap, the Hamming distance of the error event can be uniquely decided. This can be better explained by an example. Assume the component code is the recursive $5/7$ code. The cycle length is $\tau = 3$. Since all t_i's and s_i's are multiples of τ, they are in the common form of $k\tau$. Let the input be $1 + D^{k\tau}$. Then the output is

$$\begin{aligned} (1 + D^{3k}) & \frac{1 + D^2}{1 + D + D^2} \\ & = (1 + D^3 + D^{2\cdot 3} + \cdots + D^{3(k-1)}) \\ & \quad \times (1 + D^3) \frac{1 + D^2}{1 + D + D^2} \\ & = (1 + D^3 + D^{2\cdot 3} + \cdots + D^{3(k-1)}) \\ & \quad \times (1 + D + D^2 + D^3) \end{aligned} \quad (6)$$

which is the summation of k shifted versions of the output sequence with an input $1 + D^3$. The weight of the output sequence of the input $1 + D^3$ is $w_0 = 2$. (It does not include the 1's at the two ends). Then the weight of the output with the input $1 + D^{3k}$ is $2 + w_0 k$. The total output weight for the error event is

$$6m + \left(\frac{\sum |t_i|}{\tau} + \frac{\sum |s_i|}{\tau} \right) w_0 \quad (7)$$

which contains the weight in both recursive component codes ($2m + \frac{\sum |t_i|}{\tau} w_0$ or $2m + \frac{\sum |s_i|}{\tau} w_0$) and the weight of the systematic bits ($2m$). For other component codes, only the term w_0 is different.

When some of the first-order error events overlap, the corresponding Hamming distance is smaller than computed using (7) and can only be computed when all x_i, t_i, and s_i are known. However, when m is small, the number of such error events is small and can be ignored for the same reason that we ignore the boundary effect. In the following part of the paper, we will just use (7) to compute the Hamming distance of an error pattern.

B. Search for Good Interleavers Using Effective Free Distance

Although for deterministic interleavers, large d_{ef} does not guarantee a good performance, small d_{ef} is usually associated with a bad performance. We can use d_{ef} as a simple criterion to rule out bad permutation polynomials.

In this subsection, we are interested in constructing interleavers that do not map a short input weight 2 error event in one component code to another short input weight 2 error event in the other component code.

For random interleavers and quadratic interleavers, we have no control over the input weight 2 error events. We can at best state that the probability for such error events to appear goes to zero as the interleaver size goes to infinity. For S-random interleavers, by definition, input weight 2 error events with length less than S in both component codes can be avoided. Typically, an S-random interleaver with $S = \sqrt{N/2}$ can be found with reasonable complexity [2].

When $t \leq S$, an S-random interleaver will map $(x, x+t)$ to (y, z) where $|y - z| > S$. But for a given component code, there are input weight 2 error events only for some values of t, namely, those t that are multiples of the cycle length. Thus, in this sense, some of the ability of S-random interleaver is wasted. Using a permutation polynomial-based interleaver, we can select coefficients such that we only avoid input weight 2 error events tailored for a given component code. In this way, we may break even larger input weight 2 error events.

Let us restrict ourselves to polynomials of the form $P(x) = bx^2 + ax$. By Corollary 2.4, $P(x) = bx^2 + ax$ is a permutation polynomial if none of the elements in vector o_b (order of b) is zero, and the vector o_a (order of a) is an all-zero vector.

Let $t + 1$ be the length of the input weight 2 error event in the first component code. Then t must be a multiple of the cycle length τ. Let the order of t be o_t. Then the length of input weight 2 error event in the other component code minus 1 is

$$\Delta(x, t) = P(x + t) - P(x) = 2btx + bt^2 + at. \quad (8)$$

The coefficient of x is $c_1 = 2bt$, which, by Property 2.9, has order $o_{c_1} = o_2 + o_b + o_t$. For $x \in \{0, 1, 2, \ldots, N-1\}$, the first term of (8) is $k \cdot p_N^{o_{c_1}}$ where $k = 0, 1, \ldots, p_N^{o_N - o_{c_1}} - 1$. Each of these values is taken $p_N^{o_{c_1}}$ times. If we plot this term with respect to x, we have $p_N^{o_N - o_{c_1}}$ horizontal lines. The last two terms in (8) are not related to x. So they provide an *offset* to the horizontal lines. This justifies the plot of $\Delta(x,t)$ in the form of Fig. 2(d).

To avoid short input weight 2 error events, when t is a small multiple of τ, we prefer all $\Delta(x,t)$ that are also multiple of τ to be far away from zero. In order to achieve this, we require the vector o_{c_1} to be relatively large, so that there are fewer horizontal lines in the $\Delta(x,t)$ plot and we will be able to select the coefficients to move the $\Delta(x,t)$ far away from zero. Since o_{c_1} is relatively large, we are only interested in the first line of $\Delta(x,t)$ either above or below zero. Their distances to zero can be represented by

$$\begin{aligned} s &= \pm\Delta(x,t) \bmod p_N^{o_{c_1}} \\ &= \pm(bt^2+at) \bmod p_N^{o_{c_1}}. \end{aligned} \tag{9}$$

Given a, b, and τ, we define $L_{a,b,\tau} = \min(|t|+|s|)$, where t and s are multiples of τ, and use it as a criterion to select good interleavers. For a given component code, d_{ef} can be computed from $L_{a,b,\tau}$. However, in the search for permutation polynomial-based interleaver based on comparing effective free distances, the cycle length is the only information we need to know about the component code.

Before doing a search for good a and b, it is useful to limit their ranges. This can be achieved by using the following two lemmas.

Lemma 4.1: For input weight 2 error event analysis, if we write b as $b = b_1 b_0 = b_1 \cdot p_N^{o_b}$, we can assume $b_1 = 1$.

Proof: From the assumption, b_1 is coprime to N. By (9), for permutation polynomial $P_1(x) = p_N^{o_b}x^2 + ax$, (9) becomes

$$s_1 = p_N^{o_b}t^2 + at \bmod p_N^{o_b+o_t+o_2}.$$

For another permutation polynomial $P_2(x) = b_1 p_N^{o_b}x^2 + ax$, (9) becomes

$$s_2 = b_1 p_N^{o_b}t^2 + at \bmod p_N^{o_b+o_t+o_2}.$$

Compute $s_2 - s_1$

$$s_2 - s_1 = (b_1 - 1)p_N^{o_b}t^2 \bmod p_N^{o_b+o_t+o_2}. \tag{10}$$

If 2 is a factor of N, since b_1 is coprime to N, b_1 is odd and $b_1 - 1$ is even. Then the right-hand side of (10) has an order at least $o_2 + o_b + 2o_t$. If 2 is not a factor of N, the right-hand side of (10) has an order at least $o_b + 2o_t$ and the computation is modulo $o_b + o_t$. In both these cases

$$s_2 - s_1 = 0 \bmod p_N^{o_b+o_t+o_2}.$$

This implies that $P_1(x)$ and $P_2(x)$ have the same input weight 2 error events, only their locations are different. From this point of view, these two polynomials are equivalent. □

Lemma 4.2: For input weight 2 error event analysis, given $b = b_1 \cdot p_N^{o_b}$, we only need to consider a such that $1 \le a < p_N^{o_b}$.

Proof: We will use the result of Lemma 4.1 and let $b = p_N^{o_b}$.

Let $a_0 = a \bmod p_N^{o_b+o_2}$. Then a can be written as $a = a_0 + lp_N^{o_b+o_2}$. We have

$$\begin{aligned} s &= \pm\left(bt^2 + \left(a_0 + lp_N^{o_b+o_2}\right)t\right) \bmod p_N^{o_b+o_t+o_2} \\ &= \pm(bt^2 + a_0 t) \bmod p_N^{o_b+o_t+o_2} \end{aligned} \tag{11}$$

which implies that $L_{a,b,\tau} = L_{a_0,b,\tau}$.

TABLE I
COMPARISON OF L FOR $b = 16$ WITH DIFFERENT a. TC WITH $5/7$ OR $7/5$ COMPONENT CODE

a	1	3	5	7	9	11	13	15
$L(5/7)$	12	18	12	24	24	18	12	6
$L(7/5)$	4	8	12	16	16	8	12	32

When 2 is not a factor of N, the above is sufficient for the proof. When 2 is a factor of N, from the previous proof, without loss of generality, we can assume $1 \le a < p_N^{o_b+o_2}$. Let $a_0 = p_N^{o_b+o_2} - a$. Then

$$\begin{aligned} s &= \pm(bt^2 + a_0 t) \bmod p_N^{o_b+o_2+o_t} \\ &= \pm\left(bt^2 + \left(p_N^{o_b+o_2} - a\right)t\right) \bmod p_N^{o_b+o_2+o_t} \\ &= \pm(bt^2 - a_0 t) \bmod p_N^{o_b+o_2+o_t} \\ &= \pm(b(-t)^2 + a_0(-t)) \bmod p_N^{o_b+o_2+o_t} \end{aligned} \tag{12}$$

and $L_{a,b,\tau} = L_{a_0,b,\tau}$. This finishes our proof. □

For a given τ and a fixed o_b, according to the previous two lemmas, we can calculate $L_{a,b,\tau}$ and select good parameters. For example, for the $5/7$ code, the cycle length is $\tau = 3$. For the $7/5$ code, the cycle length is $\tau = 2$. For $N = 2^n$, $p_N = [2]$, and $o_N = [n]$. If we fix $o_b = [4]$, which corresponds to $b = 16$, the search results are shown in Table I.

For a frame size of 1024, simulation results for TCs with $5/7$ component code and $7/5$ component code are shown in Figs. 4 and 5, respectively. The orderings of the performance are the same as given by the $L_{a,b,\tau}$ criterion.

Once o_b is fixed, it seems that the effective free distance is a simple way to select good a and b for the permutation polynomial. However, the effective free distance alone does not provide enough information to select o_b. For example, it can be seen from (9) that the larger the o_b, the better the opportunity to select a good a to achieve larger effective free distances. But from computer simulations, it can be observed that the best performance of the permutation polynomial increases with o_b up to a certain degree. Then it starts to decrease. In order to explain this and find more accurate ways to select the parameters, we need to investigate higher input weight error events.

C. Higher Input Weight Error Events

We know that input weight $2m$ error events are important for permutation polynomial-based interleavers. In order to find the minimum distance of this subset of error events, we need to check these error events for small m. (Large m is usually associated with large Hamming distance as shown in (7).)

As mentioned in the previous sections, for an input weight $2m$ error event, among the $3m$ variables (x_i, t_i, and s_i for $i = 1, \ldots, m$) that describe the error event, $2m$ of them are independent. One way to find the error event is first to fix the error pattern $[t_1, \ldots, t_m, s_1, \ldots, s_m]$ and then enumerate x_1. Given the error pattern and x_1, all other x_i can be computed using any $m-1$ out of m equations in (3). Finally, these $3m$ values can be applied to the unused equation in (3) or (4) to verify if they form a valid error event.

Since permutation polynomial-based interleavers are highly structured, we show next that it is not necessary to enumerate

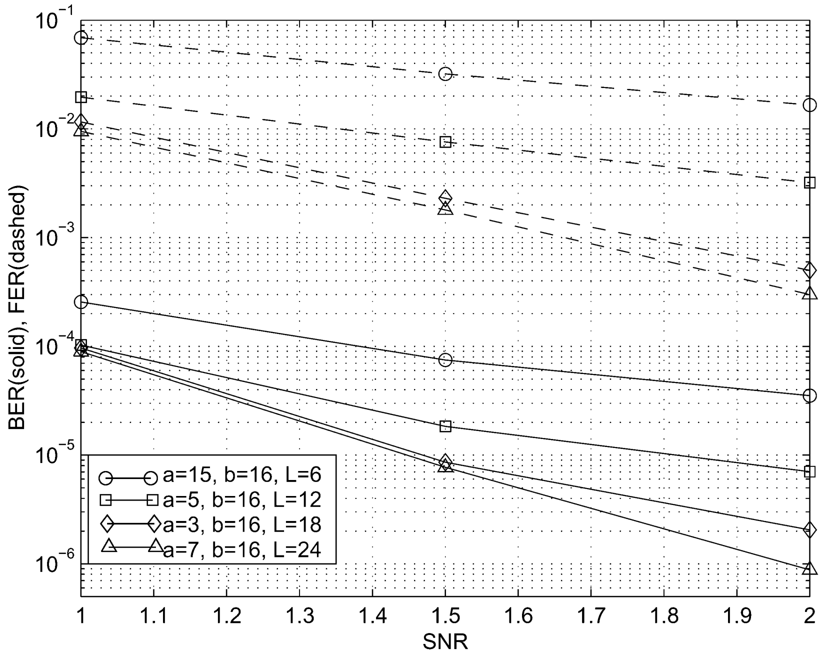

Fig. 4. TC with $5/7$ component code. Frame size 1024. Second degree permutation polynomial-based interleaver (fix $b = 16$, different a).

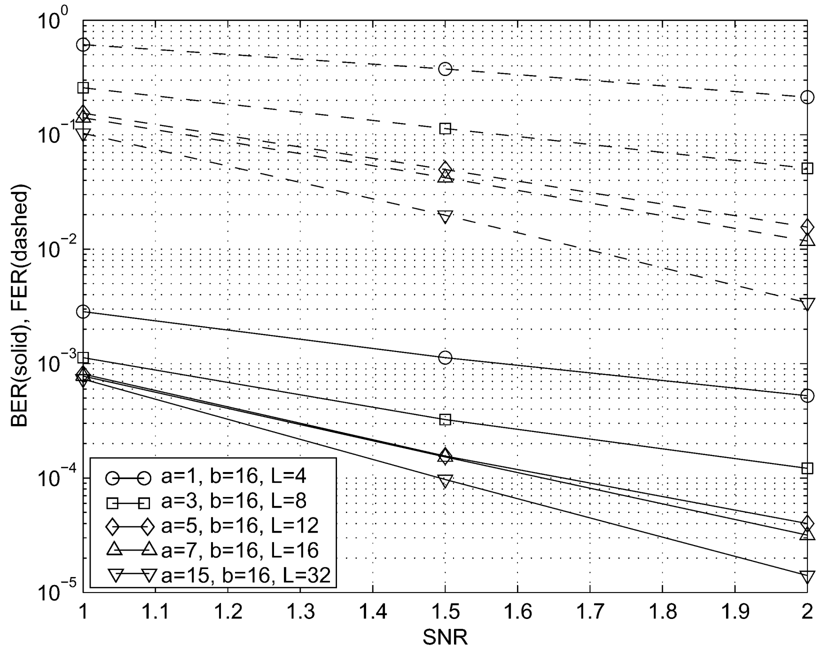

Fig. 5. TC with $7/5$ component code. Frame size 1024. Second degree permutation polynomial-based interleaver (fix $b = 16$, different a).

all x_1 from 0 to $N-1$. On the contrary, checking the range from 0 to $p_N^{o_N-o_2-o_b}-1$ is enough. Assume we have an error event as shown in Fig. 3. Each equation in (3) has the form

$$P(y_1)-P(y_2)=\left(by_1^2+ay_1\right)-\left(by_2^2+ay_2\right)=s. \quad (13)$$

In the first component code, if we cyclicly shift all $2m$ endpoints of the m first-order error events by any number T, we still have a valid input weight $2m$ error event under the tail-biting assumption. On the other side of the interleaver, after the cyclic shift, (13) becomes

$$\begin{aligned}P(y_1+T)-P(y_2+T)&=(by_1^2+ay_1)-(by_2^2+ay_2)\\&\quad+2bT(y_1-y_2)\\&=s+2bT(y_1-y_2). \quad (14)\end{aligned}$$

When $2bT=0 \bmod N$

$$P(y_1+T)-P(y_2+T)=s \quad (15)$$

and the interleaved points in the second component code also form a valid input weight $2m$ error event. The smallest T that satisfies $2bT=0 \bmod N$ is $T=p_N^{o_N-o_2-o_b}$. To summarize, the input weight $2m$ error events have a cyclic structure. If we cyclicly shift all $2m$ endpoints of the error event in the first component code by multiples of $p_N^{o_N-o_2-o_b}$, we will still have a valid input weight $2m$ error event for the entire TC. Thus, in searching for error events, it is sufficient to enumerate all x_1 from 0 to $p_N^{o_N-o_2-o_b}-1$. When $p_N^{o_N-o_2-o_b}$ is small, this method is efficient.

For large frame sizes, $p_N^{o_N-o_2-o_b}$ is usually large, making it time-consuming to enumerate x_1. It is interesting to directly solve for x_1 given the error pattern $[t_1,\ldots,t_m,s_1,\ldots,s_m]$. In order to do this, we prefer to write the constraints of the input weight $2m$ error event with the given error pattern into a single equation.

In Section IV-A, an input weight $2m$ error event is defined by (5) and $m-1$ equations in (3). Since we fix t_i and s_i, there are m nonlinear equations and m variables $(x_i, i=1,\ldots,m)$. However, in order to solve x_1 directly, we would like to transform these m equations to a single equation with only one variable, x_1. In other words, we need to apply $m-1$ equations in (3) to (5) and cancel x_i, $i=2,\ldots,m$. In order to do this, we need to be able to solve the following problem: given N, a, b, and s, if $P(y)-P(x)=s$, write y as a function of x.

Before proceeding, we need some more definitions. Given a permutation polynomial $P(x)=ax+bx^2$ and a fixed s, define a sequence $\{y_i\}$ such that

$$\begin{aligned} P(y_0)-P(0)&=s\\ P(y_1)-P(1)&=s\\ P(y_2)-P(2)&=s\\ P(y_3)-P(3)&=s\\ &\vdots \end{aligned} \tag{16}$$

Then recursively define $\Delta_0(i)\triangleq y_i-i$, $\Delta_1(i)\triangleq\Delta_0(i+1)-\Delta_0(i)$, $\Delta_2(i)\triangleq\Delta_1(i+1)-\Delta_1(i)$, etc. Note that all y_i and $\Delta_k(i)$ are a function of a, b, and s.

We have the following theorem.

Theorem 4.3: Given N, $P(x)$, and s, $\Delta_k(i)$ has the same order for all i. If the order of $\Delta_k(i)$ is denoted as o_{Δ_k}, then $o_{\Delta_0}=o_s$ and $o_{\Delta_k}=o_{\Delta_{k-1}}+o_b+o_{2k}$ for $k>0$, which implies that

$$o_{\Delta_k}=ko_b+ko_2+\sum_{n=1}^{k}o_n+o_s,\quad \text{for all } k.$$

The proof for this theorem is given in the Appendix.

Since the order of Δ_k is strictly increasing with respect to k, it will eventually be larger than o_N. If K is the largest number such that $o_{\Delta_K}\not\geq o_N$, then $\Delta_{K+1}(i)=0 \bmod N$. By the definition of $\Delta_{K+1}(i)$, $\Delta_K(i)$ is a constant for all i's. This result is summarized in the following corollary.

Corollary 4.4: For given N, $P(x)$, and s, if K is the largest number such that $o_{\Delta_K}\not\geq o_N$, then $\Delta_K(i)=\Delta_K$ is a constant for all i.

Note that K is a function of N, a, b, and s. When the parameters are not clear from the context, we will explicitly write K as a function, such as $K(s)$.

If the K in Corollary 4.4 is found, for all $k>K$, $\Delta_k(i)=0$. Furthermore, if for all $0\leq k\leq K$, the $\Delta_k(0)$ is known, all $\Delta_k(i)$ can be computed for all i using the definitions of $\Delta_k(i)$. For example, $\Delta_K(i)=\Delta_K(0)$, $\Delta_{K-1}(i)=\Delta_{K-1}(0)+i\Delta_K(0)$, etc. For the simplicity of the expressions, we will use Δ_k to represent $\Delta_k(0)$ when the context is clear. When s needs to be specified, we will use the notation $\Delta_k(s)$.

Now we have the tool to find the relationship between x and y when we know $P(y)-P(x)=s$. This is summarized in the following theorem.

Theorem 4.5: For given N, $P(x)$, and s, if $P(y)-P(x)=s$, then

$$y=x+\Delta_0(s)+x\Delta_1(s)+\frac{x(x-1)}{2!}\Delta_2(s)+\frac{x(x-1)(x-2)}{3!}\Delta_3(s)+\cdots. \tag{17}$$

We provide a sketch for a proof of this theorem. For the parameters given, we can compute all $\Delta_k(0)$ for $k=0,\ldots,K(s)$. From Corollary 4.4, we know $\Delta_K(i)$ is a constant. Then we can recursively compute $\Delta_{K-1}(i)$, $\Delta_{K-2}(i)$, etc., using their definitions. All these computations are a summation of a series. Finally, we can compute $\Delta_0(i)$ for all i. Then by the definition of $\Delta_0(i)$, $y_x=x+\Delta_0(x)$.

If we define $\binom{x}{k}=0$ when $x<k$, then (17) can be written as

$$y=F(x,s)\triangleq x+\sum_{k=0}^{\infty}\binom{x}{k}\Delta_k(s). \tag{18}$$

Since $\binom{x}{k}$ is always an integer, $F(x,s)$ will always be integer valued. However, it may not be a polynomial of x. This is because $\frac{\Delta_k(s)}{k!}$ may not be an integer. Since we do not know much about polynomials with fractional coefficients, it is preferred to somehow transform $F(x,s)$ to an integer valued polynomial.

Given N, define $D(k)=\frac{k!}{p_N^{o_2+\cdots+o_k}}$. $D(k)$ can be written as

$$D(k)=\prod_{i=2}^{k}\frac{i}{p_N^{o_i}}.$$

Each term of $D(k)$ is in the form of $\frac{i}{p_N^{o_i}}$. Since $p_N^{o_i}=\gcd(N,i)$, $\frac{i}{p_N^{o_i}}$ is also an integer and it is either equal to 1 or coprime to N. The order of $\frac{i}{p_N^{o_i}}$ is always a zero vector. Then the product $D(k)$ is also an integer and it is either 1 or is coprime to N. If we multiply $\frac{\Delta_k}{k!}$ by $D(k)$, then we have $\frac{\Delta_k}{p_N^{o_2+o_3+\cdots+o_k}}$. Since

$$o_{\Delta_k}=ko_b+ko_2+\sum_{n=1}^{k}o_n+o_s$$

it has a factor of $p_N^{o_2+o_3+\cdots+o_k}$ and $D(k)\frac{\Delta_k}{k!}$ is an integer and its order is $ko_b+ko_2+o_s$. Next consider $D(K)\frac{\Delta_k}{k!}$ for $K>k$. Since $D(K)=D(k)\prod_{i=k+1}^{K}\frac{i}{p_N^{o_i}}$

$$D(K)\frac{\Delta_k}{k!}=D(k)\frac{\Delta_k}{k!}\prod_{i=k+1}^{K}\frac{i}{p_N^{o_i}}. \tag{19}$$

As we have shown previously, both

$$D(k)\frac{\Delta_k}{k!}\quad\text{and}\quad\prod_{i=k+1}^{K}\frac{i}{p_N^{o_i}}$$

are integers. Then $D(K)\frac{\Delta_k}{k!}$ for $K>k$ is also an integer. Its order is still $ko_b+ko_2+o_s$.

Using this result, it can be easily seen that $D(K)F(x,s)$ is a polynomial of x with integer coefficients. This property will be used in the following parts.

1) Finding the Error Event by Solving Equations: Using Theorem 4.5, given an input weight $2m$ error pattern $[t_1,t_2,\ldots,t_m,s_1,s_2,\ldots,s_m]$, searching for error events with the error pattern can be transformed into solving a polynomial equation.

An error event can be formed if (4) holds. Applying the result of Theorem 4.5 to the first $m-1$ equations in (3), we have

$$\begin{aligned} x_2 &= F(x_1,s_1)\\ x_3 &= F(x_1+t_1,s_2)\\ x_4 &= F(x_2+t_2,s_3)\\ x_5 &= F(x_3+t_3,s_4)\\ &\vdots\\ x_m &= F(x_{m-2}+t_{m-2},s_{m-1}). \end{aligned} \tag{20}$$

Applying (20) in (4), in general, we have a polynomial equation of x_1. For a general polynomial equation, the complexity to find the solutions or even find the number of solutions is high [13]. However, we have simpler methods to deal with polynomials associated with input weight $2m$ error events for small m. In this subsection, we will only consider $m=1,2$, and 3.

The input weight 2 error pattern is the simplest case. Here (20) is not used and (4) reduces to

$$2bx_1+bt_1^2+at_1-s_1=0. \tag{21}$$

This is a linear polynomial of x_1 in the form

$$c_1x+c_0=0. \tag{22}$$

We know that (22) has a solution if and only if $o_{c_0}\geq o_{c_1}$. If this condition is satisfied, we can divide (22) by $p_N^{o_{c_1}}$. The result is a first-degree permutation polynomial modulo $p_N^{o_N-o_{c_1}}$ and there is a unique solution. If x_0 is the solution modulo $p_N^{o_N-o_{c_1}}$, all $x_0+kp_N^{o_N-o_{c_1}}$ for any k are solutions to (22) modulo N. Then, for (22), there are exactly $p_N^{o_{c_1}}$ solutions. In most cases, we are only interested in the Hamming distance of the error event and the multiplicity for such a Hamming distance. It is not necessary to find out the exact location of the error event. So instead of solving (22), what we need to do is just check if $o_{c_0}\geq o_{c_1}$. If true, then we add the multiplicity of $p_N^{o_{c_1}}$ to the corresponding spectrum line. The Hamming distance of the error event is directly decided by its error pattern using (7).

Equation (4) is more complex for the $m>1$ cases. However, we can show that, under some mild conditions, at least for $m=2$ and $m=3$, (4) can still be transformed to a permutation polynomial when (20) is applied. For larger m, we conjecture that the same is also true. Here, we do not consider larger m cases in our minimum distance computation for they usually correspond to larger Hamming distances. We will satisfy ourselves by proving the cases for input weights 4 and 6.

For the input weight 4 case, using (20), (4) becomes

$$\begin{aligned} 2b\left[x_1t_1-\left(x_1+\sum_{k=0}^{\infty}\binom{x_1}{k}\Delta_k(s_1)\right)t_2\right]&\\ +b\left(t_1^2-t_2^2\right)+a(t_1-t_2)-(s_1-s_2)=0.& \end{aligned} \tag{23}$$

Collecting terms with respect to x_1, it becomes

$$\begin{aligned} &\left(-2b\Delta_0(s_1)t_2+b\left(t_1^2-t_2^2\right)+a(t_1-t_2)-(s_1-s_2)\right)\\ &\quad+2b(t_1-t_2-\Delta_1(s_1)t_2)x_1\\ &\quad-2bt_2\sum_{k=2}^{\infty}\frac{\Delta_k(s_1)}{k!}\prod_{m=0}^{k-1}(x_1-m)=0. \end{aligned} \tag{24}$$

This is a polynomial of x_1 with possibly fractional coefficients. For s_1, we can find the K in Corollary 4.4. Then multiply (24) by $D(K)$. From the analysis in the previous part, the result is a polynomial with integer coefficients in the form of

$$c_0+c_1x_1+c_2(x_1)=0 \tag{25}$$

where

$$\begin{aligned} c_0 &= \Big(-2b\Delta_0(s_1)t_2+b\left(t_1^2-t_2^2\right)+a(t_1-t_2)\\ &\qquad -(s_1-s_2)\Big)D(K)\\ c_1 &= 2b(t_1-t_2-\Delta_1(s_1)t_2)D(K)\\ c_2(x) &= -2bt_2\sum_{k=2}^{\infty}\frac{\Delta_k(s_1)D(K)}{k!}\prod_{m=0}^{k-1}(x_1-m). \end{aligned} \tag{26}$$

c_0 and c_1 are integers and $c_2(x_1)$ is a polynomial of x_1. Furthermore, since $D(k)$ is either 1 or is coprime to N, multiplying by $D(k)$ will not change the solutions for (24).

Now, let us check the order of terms in (26). We have shown that the order of $\frac{\Delta_k(s_1)}{k!}D(K)$ is $ko_b+ko_2+o_{s_1}$ for $K\geq k$. If we expand the third term of $c_2(x_1)$ to a polynomial of x_1, all the coefficients should have an order at least $3o_b+3o_2+o_{s_1}+o_{t_2}$. If $t_1=t_2$, $o_{c_1}=2o_b+2o_2+o_{s_1}+o_{t_2}$. Since all the elements in o_b are positive, o_{c_1} is strictly less than the order of the orders of all terms in $c_2(x_1)$. By Corollary 2.5, the polynomial of x_1 formed by $\frac{c_1x_1+c_2(x_1)}{p_N^{o_{c_1}}}$ is a permutation polynomial modulo $p_N^{o_N-o_{c_1}}$. In order to apply Corollary 2.5 to $t_1\neq t_2$ case, we require that

$$o_{c_1}\ll 3o_b+3o_2+o_{s_1}+o_{t_2}. \tag{27}$$

Since we are interested in small Hamming distance error events, t_1 and t_2 are usually small, which implies that t_1-t_2 is also small. The above condition is usually satisfied.

Given an error pattern $[t_1,t_2,s_1,s_2]$, first we need to verify that (27) holds. (In our search, we have never observed a case when this does not hold). Then similar to the input weight 2 case, we only need to check if $o_{c_0}\geq o_{c_1}$ for the error pattern. If it is true, $p_N^{o_{c_1}}$ error events with this error pattern exist. If we are only interested in the spectrum, we do not need to solve the equation and only $\Delta_0(s_1)$ and $\Delta_1(s_1)$ need to be computed for the error pattern.

For the input weight 6 error event, we can use the same procedure as for the input weight 4 error event analysis. Equation (4) will be finally reduced to the same form as in (25). Similar to the input weight 4 case, a factor D needs to be multiplied to the equation to transform it into a polynomial with integer coefficients. Here $D=D(\max(K(s_1),K(s_2)))$. Then in (25)

$$c_1=2b(t_1-t_2+t_3-\Delta_1(s_1)t_2+\Delta_1(s_2)t_3)D$$

and the coefficients of $c_2(x)$ have orders at least equal to

$$min(3o_b+3o_2+o_{s_1}+o_{t_2},3o_b+3o_2+o_{s_2}+o_{t_3}).$$

The condition to apply Corollary 2.5 is that

$$o_{c_1} \ll \min(3o_b+3o_2+o_{s_1}+o_{t_2}, 3o_b+3o_2+o_{s_2}+o_{t_3}). \quad (28)$$

This is usually the case when t_i's are small. Then $\frac{c_1x_1+c_2(x_1)}{p_N^{o_{c_1}}}$ is a permutation polynomial modulo $p_N^{o_N-o_{c_1}}$. As in the input weight 4 case, we can compare o_{c_0} and o_{c_1} to see if error events with such an error pattern exist and find the multiplicities of the error events. If one is only interested in the spectrum, there is no need to solve the equation and only $\Delta_0(s_1)$, $\Delta_0(s_2)$, $\Delta_1(s_1)$, $\Delta_1(s_2)$ need to be computed for the given error pattern.

2) Upper Bounds on o_b: From input weight 2 error event analysis, we have the result that, given N, the larger the o_b, the larger the flexibility we have to choose good interleavers and the better the performance. But from simulations, we can observe that the best possible performance for permutation polynomial interleaver grows with o_b up to a certain value. Then the performance degrades again. We can use Theorem 4.5 to find some simple upper bounds for o_b.

a) Upper bound on o_b from input weight 4 error events: We start from (25). When $m = 2$, assuming condition (27) holds, then when $o_{c_0} \geq o_{c_1}$, if a solution of x_1 exists for (25), an error event exists for the given error pattern starting at the x_1.

One special case is that, for some error pattern, every x_1 from 0 to $N-1$ is a solution. This will lead to a spectrum line with high multiplicity, which may dominate the performance even if it is not the minimum distance. Let us consider $t_1 = t_2 = t$ and $s_1 = s_2 = s$. Then c_0 and c_1 in (26) become $-2b\Delta_0(s)tD(K)$ and $-2b\Delta_1(s)tD(K)$, respectively. It is easy to see that $o_{c_1} \geq o_{c_0}$. All coefficients in $c_2(x_1)$ have a larger order than o_{c_1}. If $o_{c_0} = o_N$ (which contains the case that some elements in o_{c_0} are larger than the corresponding elements in o_N numerically), then (25) becomes an all-zero polynomial. Trivially, every possible x_1 is a solution to the polynomial.

This special case places an upper bound for o_b such that we require that o_N has at least one element larger than the corresponding element in $o_{c_0} = o_b + o_2 + o_s + o_t$.

An interesting special case exceeding this upper bound is $o_b = o_N$, which implies that $b = 0 \bmod N$. The permutation polynomial reduces to a linear interleaver. Now the error event can be described as

$$a(t_1 - t_2) = s_1 - s_2. \quad (29)$$

It is easy to see that when $t_1 = t_2$ and $s_1 = s_2$, (29) trivially holds. Obviously, every x_1 is a solution. This is exactly why a linear interleaver has many input weight 4 error events and cannot work well at medium to long frame sizes.

b) Upper bound on o_b from input weight 6 error events: For the input weight 6 error event case, we also start from (25). Assuming the condition (28) holds, then

$$\begin{aligned} C_0 = \Big[&2b[t_3\delta_0(s_2) - t_2\delta_0(s_1)] + 2bt_3t_1\delta_1(s_2) \\ &+ 2bT_1T_3 + b\left(t_1^2 - t_2^2 + t_3^2\right) \\ &+ a(t_1 - t_2 + t_3) - (s_1 - s_2 + s_3)\Big]D \quad (30) \end{aligned}$$

$$c_1 = [2b(t_1 - t_2 + t_3) + 2b(t_3\delta_1(s_2) - t_2\delta_1(s_1))]\,D \quad (31)$$

where $D = D(\max(k(s_1), k(s_2)))$, and the coefficients of $c_2(x_1)$ have higher order than o_{c_1}. We are still interested in the error pattern that every x_1 is a solution.

We discovered that the error patterns of the form $[2t, t, -t, s, -s, -2s]$ are critical. (Since we require $t_1 \geq |t_2|$ and $t_1 \geq |t_3|$ to avoid some double counting, many other critical error patterns are equivalent to this one). The error pattern that corresponds to the smallest Hamming distance is when $t = \tau$ and $s = \pm t$.

For this error pattern, we have $t_1 - t_2 + t_3 = s_1 - s_2 + s_3 = 0$. Then c_0 and c_1 reduce to

$$c_0 = \left[-2bt[\Delta_0(-s) + \Delta_0(s)] - 4bt^2\Delta_1(-s)\right]D \quad (32)$$

$$c_1 = -2bt(\Delta_1(-s) + \Delta_1(s))D. \quad (33)$$

In order to proceed, we need two lemmas.

Lemma 4.6: $\Delta_0(-s) + \Delta_0(s)$ has order $o_b + o_2 + 2o_s$.

Proof: Let $P(y) = s$ and $P(x) = -s$, where $P(x) = bx^2 + ax$. By the definition of $\Delta_0(i)$, we know $y = \Delta_0(s)$ and $x = \Delta_0(-s)$. Define $c = y + x$, then $y = c - x$. Since $P(y) + P(x) = s + (-s) = 0$, we have

$$b(c-x)^2 + a(c-x) + bx^2 + ax = 0. \quad (34)$$

Rewriting it as a polynomial of c, we have

$$bc^2 + (-2bx + a)c + 2bx^2 = 0 \quad (35)$$

which is a permutation polynomial of c and the solution for c has the same order as the term $2bx^2$. The order of x, or $\Delta_0(-s)$, is known by Theorem 4.3 to be o_s. Thus, c has an order $o_b + o_2 + 2o_s$. □

Lemma 4.7: $\Delta_1(-s) + \Delta_1(s)$ has an order at least $o_b + 2o_2 + 2o_s$.

Proof: Assume we have y_0, y_1, x_0, and x_1 such that

$$P(y_0) = by_0^2 + ay_0 = s, \quad (36)$$

$$P(y_1) - P(1) = by_1^2 + ay_1 - b - a = s \quad (37)$$

$$P(x_0) = bx_0^2 + ax_0 = -s \quad (38)$$

$$P(x_1) - P(1) = bx_1^2 + ax_1 - b - a = -s. \quad (39)$$

Then $\Delta_1(s) = y_1 - y_0 - 1$ and $\Delta_1(-s) = x_1 - x_0 - 1$. Define $c \triangleq y_0 + x_0$ and $d = \Delta_1(s) + \Delta_1(-s) = y_1 + x_1 - c - 2$. Then $y_1 = d + c - x_1 + 2$. Applying this in (37) and (39), we have

$$\begin{aligned} bd^2 + (-2bx_1 + 2bc + 4b + a)d + (bc^2 - 2bc(x_1 - 1) \\ + 2bc + 2b(x_1 - 1)^2 + ac) = 0. \quad (40) \end{aligned}$$

This is a permutation polynomial of d. From the previous lemma, we know c has an order $o_b + o_2 + 2o_s$. Then in the constant term in (40), both $2b(x_1 - 1)^2$ and ac have an order $o_b + o_2 + 2o_s$ and the other terms have higher orders. So the order of d is at least $o_b + 2o_2 + 2o_s$. Stronger results may exist but this is sufficient for our current purpose. □

Now we return to (32). The order of c_0 and c_1 can be found using the above two lemmas. o_{c_0} is at least $2o_b + 2o_2 + 3o_\tau$ and o_{c_1} is at least $2o_b + 3o_2 + 3o_\tau$. When each element in o_N is not larger than the corresponding element in $2o_b + 2o_2 + 3o_\tau$, both c_0 and c_1 are zero modulo N and all x_1's are solutions to the equation. This can serve as another upper bound for o_b.

This upper bound in o_b is not stringent. If the error pattern corresponds to a large enough Hamming distance, then even though the multiplicity is high, the effect in the performance can be low. As shown in (7), the Hamming distance of the error event is $18+6w_0$. For example, for the recursive $5/7$ code, $w_0 = 2$. The Hamming distance is 30. For the recursive $23/35$ code, $w_0 = 3$, the Hamming distance is 36 and for the recursive $37/23$ code, $w = 8$ and the Hamming distance is 66. Sometimes, for a code with a large cycle length, this upper bound on o_b can be relaxed.

3) Range of a and b to Search: In order to search for good second-degree permutation polynomials for the interleaver, we need to select the range of a and b to be searched. It is shown in Lemmas 4.1 and 4.2 that, for input weight 2 error events, if o_b is fixed, we can pick $b = b_0 = p_N^{o_b}$ and consider a from 1 to b_0. Unfortunately, for general error events, we only have a similar result for the range of a.

Theorem 4.8: For second-degree permutation polynomial-based interleavers, we only need to consider a in the range between 1 and $2b$.

Proof: Consider $P(x) = bx^2 + ax$ and $P'(x) = bx^2 + (a + 2b)x$. Assume we have an error event of input weight m for the code using $P(x)$ based interleaver. The positions of the errors are at $x_1, x_2, \ldots, x_m$ in the first component code and $P(x_1), P(x_2), \ldots, P(x_m)$ in the second component code. Consider the code based on $P'(x)$. Assume there are errors at $x_1 - 1, x_2 - 1, \ldots, x_m - 1$ in the first component code. Then

$$P'(x_i-1) = b(x_i-1)^2+(a+2b)(x_i-1) = bx_i^2+ax_i-(a+b).$$

$P'(x_i - 1)$ is a shift of $P(x_i)$ by $a + b$. Thus, the same error event can be found in a code using both interleavers. The only difference is that the error positions are cyclicly shifted to the left by 1 in the first component code and by $a + b$ in the second component code. The spectrums of the two codes are the same. $\square$

For input weight 4 error events, in general, we have to search all b and all $1 \leq a \leq 2b$ for given o_b. This is tedious. However, it can be shown that under some conditions, we can still use the limited ranges of b given in Lemma 4.1. More precisely, we only need to consider $b = b_0 = p_N^{o_b}$. This is equivalent to showing that, under some conditions, if we replace b by b_0, the same error pattern exists in both cases with the same multiplicity.

If we replace b by b_0 in (25), it becomes

$$c_0' + c_1' x_1 + c_2'(x_1) = 0. \tag{41}$$

Since $\Delta_k(s)$'s are functions of a and b, in c_1' and c_0', $\Delta_k'(s)$ will be used instead of $\Delta_k(s)$. However, $\Delta_k'(s)$ and $\Delta_k(s)$ have the same order.

Showing that the same error pattern exists in both cases, it is equivalent to showing that (25) and (41) have the same number of solutions for the error pattern. This is further equivalent to show that $o_{c_1} = o_{c_1'}$ and $o_{c_0} \geq o_{c_1} \Leftrightarrow o_{c_0'} \geq o_{c_1'}$. One sufficient condition for this to happen is both

$$t_1 = t_2 \quad \text{or} \tag{42}$$

$$o_{t_1-t_2} \ll o_b + o_2 + o_{s_1} + o_{t_2} \tag{43}$$

and

$$s_1 = s_2 \quad \text{or} \tag{44}$$

$$o_{s_1-s_2} \ll o_b + o_2 + o_{s_1} + o_{t_2}. \tag{45}$$

TABLE II
BEST PERMUTATION POLYNOMIALS FOR DIFFERENT COMPONENT CODES. FRAME SIZE 256

Component code	Cycle length (τ)	Best polynomial	d_{min} (multiplicity)	Figure
7/5	2	$15x + 16x^2$	18(512)	6
5/7	3	$15x + 32x^2$	28(512)	7
37/21	4	$7x + 8x^2$	24(256)	8
21/37	5	$15x + 32x^2$	28(512)	9
37/25	6	$15x + 16x^2$	24(512)	10
23/35	7	$15x + 32x^2$	36(512)	11

The inequalities (43) and (45) are used to guarantee that the orders of c_0 and c_1 are dominated by $s_1 - s_2$ and $t_1 - t_2$, respectively, when $t_1 \neq t_2$ and $s_1 \neq s_2$. In the following analysis, we assume that these conditions are satisfied.

When $t_1 = t_2$, applying this to (26), we have

$$c_0 = (-2b\Delta_0(s_1)t_2 - (s_1 - s_2))D(K)$$

and

$$c_1 = -2b\Delta_1(s_1)t_2 D(K).$$

Then $o_{c_1} = 2o_b + 2o_2 + o_{s_1} + o_{t_2}$. If $s_1 = s_2$, $o_{c_0} = o_b + o_2 + o_{s_1} + o_{t_2}$. If (45) holds, $o_{c_0} = o_{s_1-s_2} \ll o_b + o_2 + o_{s_1} + o_{t_2}$. In both these cases, $o_{c_0} \ll o_{c_1}$ and no solution exists for (25). The same result can be derived for (41).

When $t_1 \neq t_2$ and (43) holds, $o_{c_1} = o_{c_1'} = o_b + o_2 + o_{t_1-t_2}$. Let us compute $c_0 - c_0'$

$$\begin{aligned} c_0 - c_0' = &-2(b\Delta_0(s_1) - b_0\Delta_0'(s_1))t_2 D(K) \\ &+(b - b_0)(t_1 - t_2)(t_1 + t_2)D(K). \end{aligned} \tag{46}$$

The two terms in (46) have orders at least $o_b + 2o_2 + o_{s_1} + o_{t_2}$ and $o_b + o_2 + o_{t_1-t_2}$, respectively. They are both higher than the o_{c_1}. Since the difference between c_0 and c_0' has higher order than o_{c_1}, $o_{c_0} \geq o_{c_1} \Leftrightarrow o_{c_0'} \geq o_{c_1'}$. Thus, we proved that b can be limited to the ranges in Lemma 4.1 when conditions in (43) and (45) hold.

For input weight 6 error patterns, a similar sufficient condition can be derived. However, the form of the sufficient condition becomes very complex and will therefore be omitted.

Although not all input weight 4 and 6 error patterns satisfy these sufficient conditions, it seems that most of the error patterns that correspond to the first few spectrum lines do. Furthermore, in our permutation polynomial search, it can be observed that extending to more general b and a usually cannot generate significantly better permutation polynomials. Therefore, in our search for the polynomials using input weight $2m$ error events spectrum, we will only consider $b = p_N^{o_b}$ and $1 \leq a < 2b$.

V. RESULTS

Given the frame size N and the component code, our search for good permutation polynomial-based interleavers is basically an enumeration of a and b for the polynomial. First we need to enumerate o_b. From the analysis in Section IV, $p_N^{o_b}$ should be relatively large but bounded by some constraints due to some special input weight 4 and 6 error events. Once we fix o_b, we let $b = p_N^{o_b}$ and enumerate all the a is in the range specified in Theorem 4.8. In this section, we will present some examples for the polynomial search for different component codes.

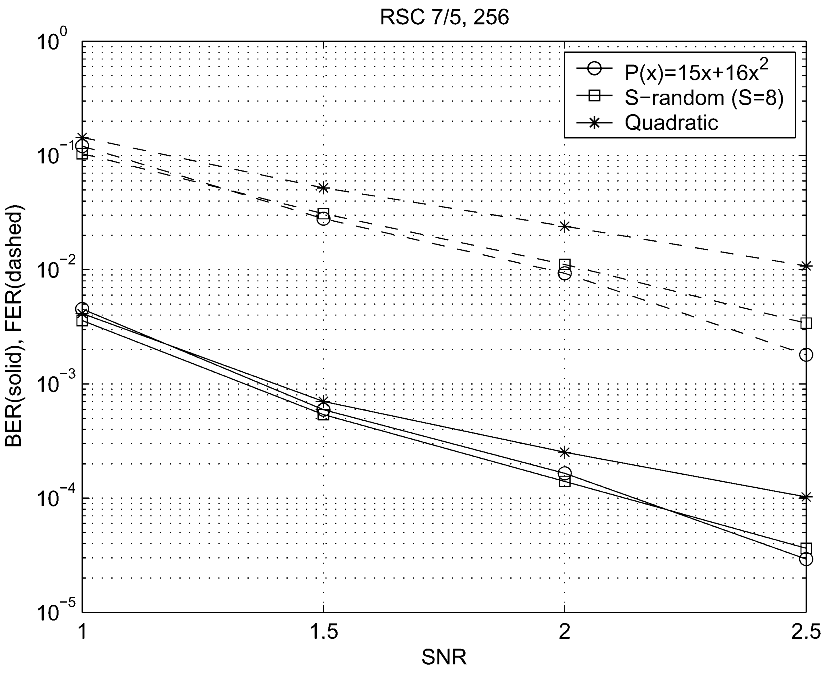

Fig. 6. TC with $7/5$ code. Frame size 256.

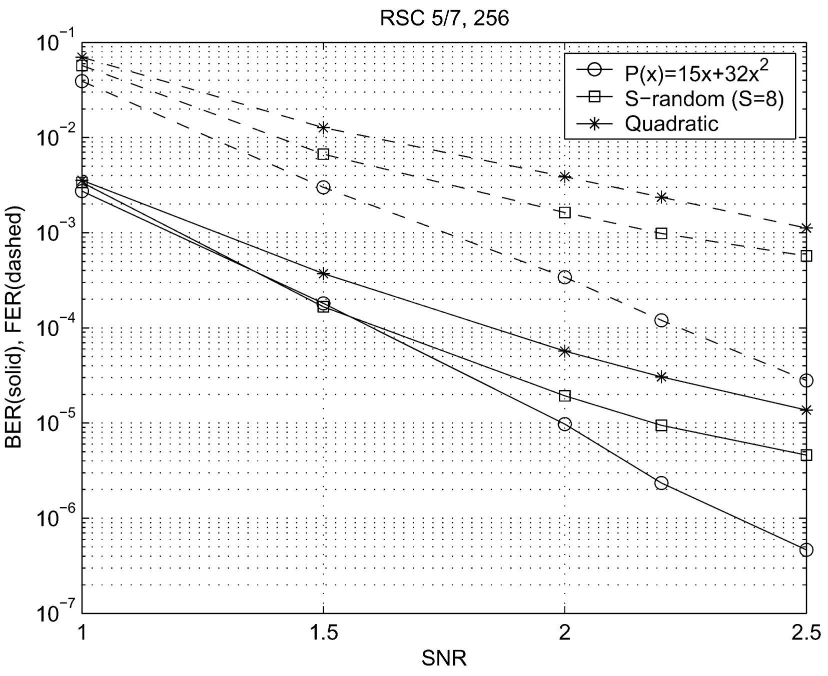

Fig. 7. TC with $5/7$ code. Frame size 256.

We consider six different systematic recursive component codes shown in the first column of Table II. Their corresponding cycle lengths are from 2 to 7, respectively. In this section, we will only consider frame sizes $N = 2^n$. Then $p_N = 2$ and all the orders are scalars.

For the frame size $N = 256$, the best permutation polynomials found for the TC using these component codes are shown in Table II. The corresponding minimum distance of the input weight $2m$ error events and its multiplicity are also shown. Computer simulation results comparing with S-random interleaver and quadratic interleaver are shown in Figs. 6–11.

In all these simulations, the permutation polynomial-based interleavers perform better than quadratic interleavers, which have a similar performance as the average of random interleavers. Comparing to S-random interleaver, the permutation polynomial-based interleavers also have a better performance for both bit-error rate and frame-error rate.

We also find the best permutation polynomial-based interleavers for TC with recursive $5/7$ component codes at frame sizes 1024 and 16384. They are based on polynomials $P(x) = 31x + 64x^2$ and $P(x) = 15x + 32x^2$, respectively. The simulation results are shown in Figs. 12 and 13. For very long frame

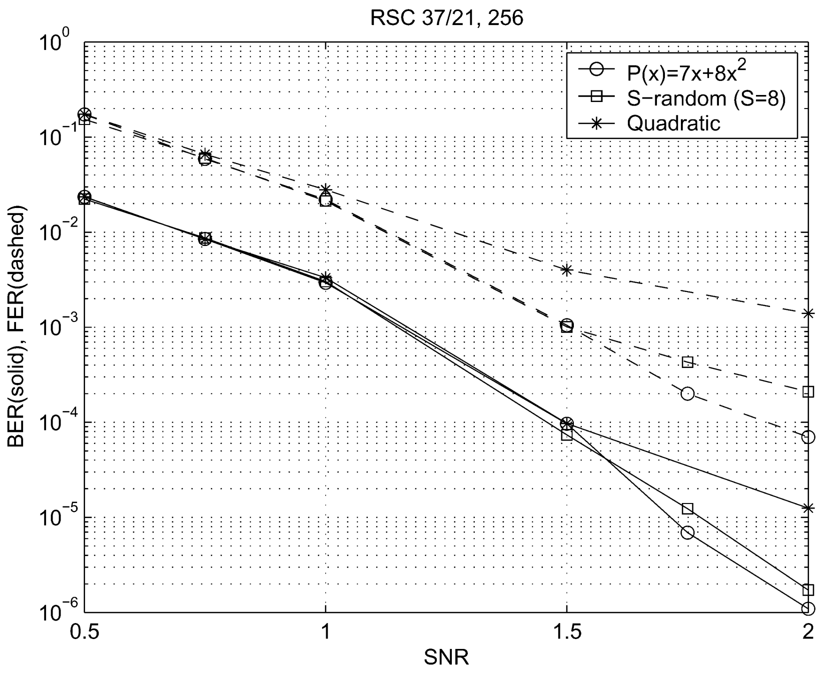

Fig. 8. TC with $37/21$ code. Frame size 256.

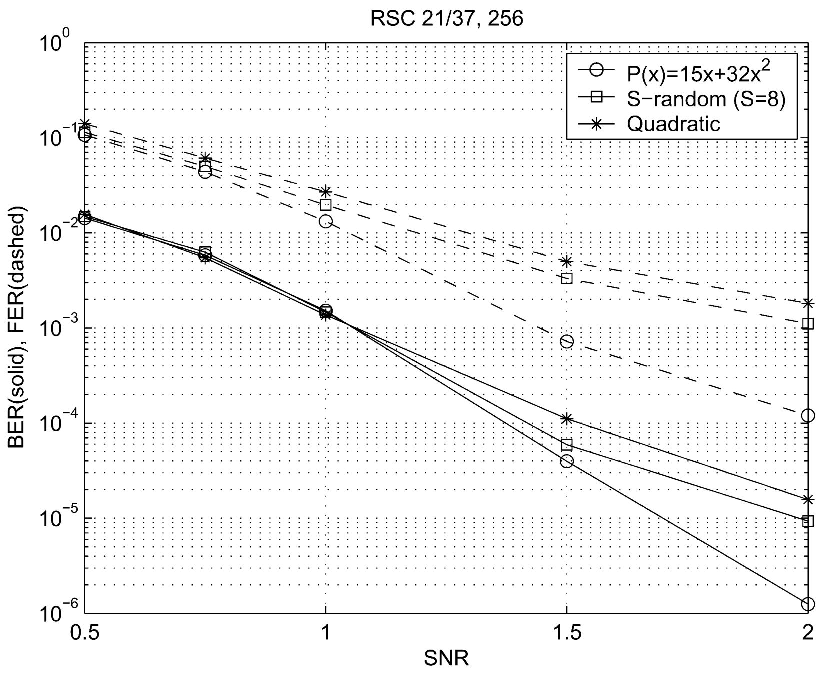

Fig. 9. TC with $21/37$ code. Frame size 256.

sizes, the performance of permutation polynomial-based interleavers are not as good as the S-random interleavers but much better than quadratic interleavers.

VI. Conclusion

In this paper, we introduced permutation polynomial-based interleavers for TCs. This is a class of deterministic interleavers, and the interleaving (de-interleaving) can be done using a simple arithmetic computation instead of a table lookup. Given the parameters of the generator polynomial for the interleaver, we can use polynomial solving to find the minimum distance (or the first few spectrum lines) of an important subset of error events, which can be used to well approximate the real minimum distance. Based on the approximated minimum distance, a limited search for good interleavers has been performed. The new interleaver is compared with the S-random interleaver and the quadratic interleaver. For short frame sizes, good interleavers that outperform the S-random interleavers have been found. For long frame sizes, the permutation polynomial-based interleavers have a performance close to the S-random interleavers. The new interleavers outperform quadratic interleavers (which are known to have a similar performance as random interleavers) for all frame sizes.

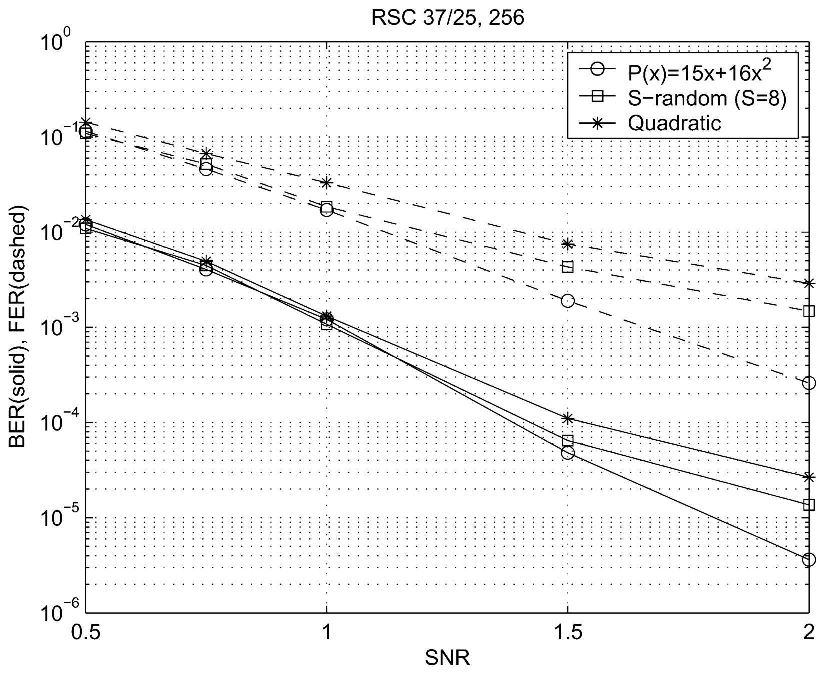

Fig. 10. TC with $37/25$ code. Frame size 256.

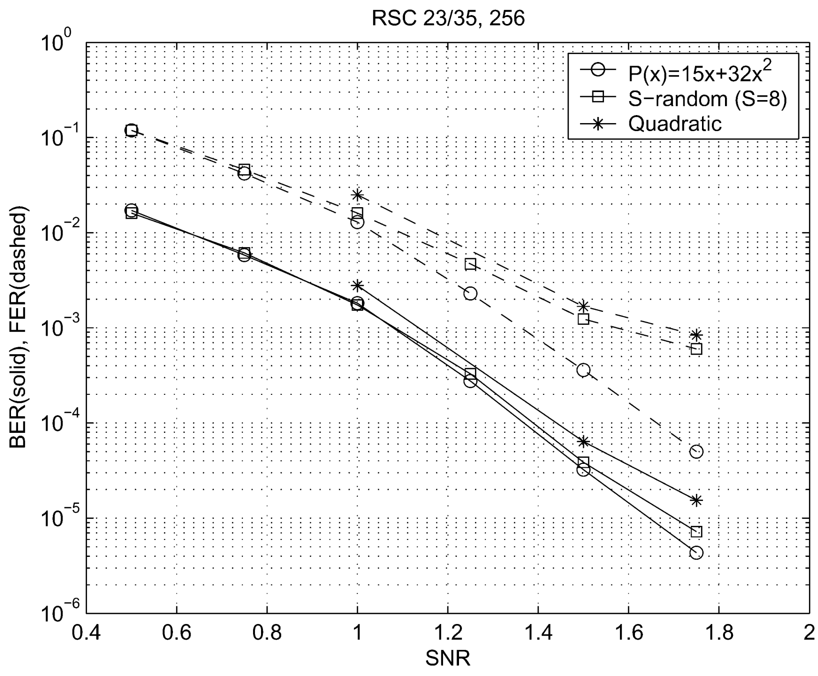

Fig. 11. TC with $23/35$ code. Frame size 256.

Appendix
Proof of Theorem 4.3

In this appendix, we prove Theorem 4.3. Induction is extensively used in the proof.

Theorem 4.3 claims that, for fixed N, $P(x)$, and s

1) $\Delta_k(i)$ has the same order for all i,
2) $o_{\Delta_0} = o_s$, and
3)

$$o_{\Delta_k} = o_{\Delta_{k-1}} + o_b + o_{2k}, \qquad \text{for } k > 0 \qquad (47)$$

which implies

$$o_{\Delta_k} = ko_b + ko_2 + \sum_{n=1}^{k} o_n + o_s, \qquad \text{for } k > 0. \qquad (48)$$

These can be proved by using induction on k.

As the first step of the induction, we need to show that $\Delta_0(m)$ has an order o_s and $\Delta_1(m)$ has an order $o_b + o_s + o_2$. By definition

$$\Delta_1(i) = \Delta_0(i+1) - \Delta_0(i)$$

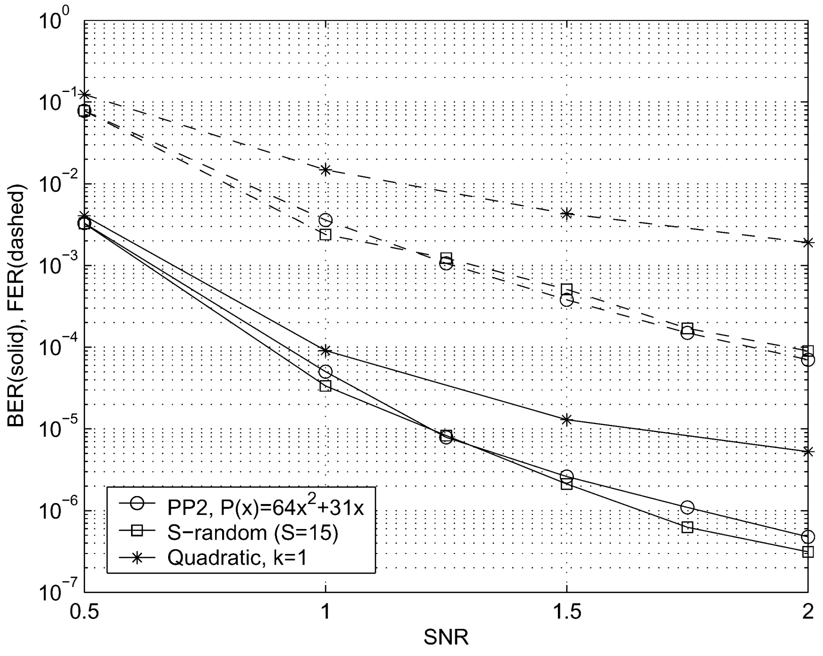

Fig. 12. TC with $5/7$ code. Frame size 1024.

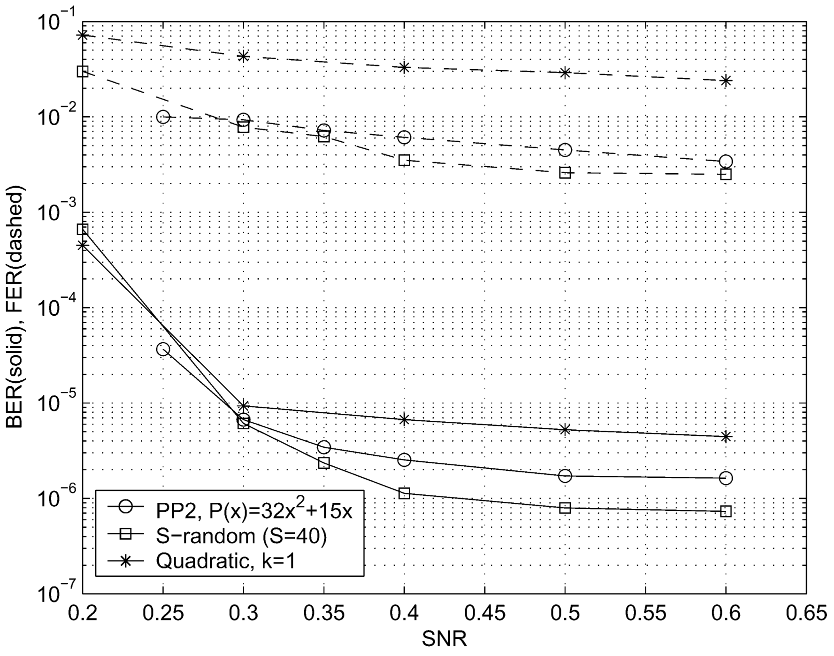

Fig. 13. TC with $5/7$ code. Frame size 16384.

$$
\begin{aligned}
&= [y_{i+1} - (i+1)] - (y_i - i) \\
&= y_{i+1} - y_i - 1
\end{aligned}
\tag{49}
$$

we have

$$
y_{i+1} = \Delta_1(i) + y_i + 1. \tag{50}
$$

Applying this to $P(y_{i+1}) - P(i+1) = s = P(y_i) - P(i)$, we have

$$
E_1(i) \triangleq (a + 2by_i + 2b)\Delta_1(i) + b\Delta_1^2(i) + 2b\Delta_0(i) = 0 \tag{51}
$$

which is a permutation polynomial of $\Delta_1(i)$ and $\Delta_1(i)$ has the same order as $2b\Delta_0(i)$. Since $\Delta_0(0) = y_0$ and $P(y_0) = ay_0 + by_0^2 = s$ is a permutation polynomial, the order of $\Delta_0(0)$ is o_s. Then by (51), the order of $\Delta_1(0)$ is $o_b + o_s + o_2$. From the definition of $\Delta_1(i)$, we have

$$
\Delta_1(i) = \Delta_0(i+1) - \Delta_0(i). \tag{52}
$$

As $\Delta_0(0)$ has an order o_s and $\Delta_1(0)$ has an order $o_b + o_s + o_2$, which is strictly larger than o_s, $\Delta_0(1)$ must also have order o_s. By alternately using (51) and (52), we can show that all $\Delta_0(i)$ have an order o_s and all $\Delta_1(i)$ have an order $o_b + o_s + o_2$.

Next, we need to find the order of $\Delta_{k+1}(i)$ using the order of $\Delta_0(i), \Delta_1(i), \ldots, \Delta_k(i)$.

Starting from (51), we can define

$$
E_k(i) \triangleq E_{k-1}(i+1) - E_{k-1}(i)
$$

and using $y_{i+1} = \Delta_1(i) + y_i + 1$, $\Delta_0(i+1) = \Delta_0(i) + \Delta_1(i)$, $\Delta_1(i+1) = \Delta_1(i) + \Delta_2(i), \ldots$, we can inductively show that $E_k(i)$ has the form of

$$E_k(i) = \left[a + 2kb + 2by_i + \sum_{j=1}^{k-1} 2g_{k,j} b\Delta_j(i) \right] \Delta_k(i) + b\Delta_k^2(i) + C(k,i) \quad (53)$$

where $C(k,i)$ is a term that contains $\Delta_1(i), \ldots, \Delta_{k-1}(i)$, but not $\Delta_k(i)$, and $g_{k,j}$'s are integers defined for $k \geq 2$ and $j = 1, \ldots, k-1$. For convenience, for all other k and j, we define $g_{k,j} = 0$.

We can verify that (51) satisfies (53) with $C(1,i) = 2b\Delta_0(i)$. For $E_{k+1}(i)$, we have

$$\begin{aligned}
&E_{k+1}(i) \\
&= E_k(i+1) - E_k(i) \\
&= \left[a + 2kb + 2by_{i+1} + \sum_{j=1}^{k-1} 2g_{k,j} b\Delta_j(i+1) \right] \Delta_k(i+1) \\
&\quad + b\Delta_k^2(i+1) + C(k,i+1) \\
&\quad - \left[a + 2kb + 2by_i + \sum_{j=1}^{k-1} 2g_{k,j} b\Delta_j(i) \right] \Delta_k(i) \\
&\quad - b\Delta_k^2(i) - C(k,i) \\
&= \Bigg[a + 2(k+1)b + 2by_i + 2b\Delta_1(i) + 2b\Delta_k(i) \\
&\quad + \sum_{j=1}^{k-1} 2g_{k,j} b(\Delta_{j+1}(i) + \Delta_j(i)) \Bigg] \Delta_{k+1}(i) \\
&\quad + b\Delta_{k+1}^2(i) + \Bigg[2b\Delta_1(i) + 2b \\
&\quad + \sum_{j=1}^{k-1} 2g_{k,j} b\Delta_{j+1}(i) \Bigg] \Delta_k(i) + C(k,i+1) - C(k,i) \\
&= \left[a + 2(k+1)b + 2by_i + \sum_{j=1}^{k} 2g_{k+1,j} b\Delta_j(i) \right] \Delta_{k+1}(i) \\
&\quad + b\Delta_{k+1}^2(i) + C(k+1,i)
\end{aligned} \quad (54)$$

where $g_{2,1} = 2$, and

$$\begin{cases} g_{k+1,1} = g_{k,1} + 1, \\ g_{k+1,j} = g_{k,j} + g_{k,j-1}, & \text{for } j = 2, 3, \ldots, k-1. \\ g_{k+1,k} = g_{k,k-1} + 1, \end{cases} \quad (55)$$

It can be verified that $g_{k,j} = \binom{k}{j}$.

The $C(k+1,i)$ term in (54) satisfies

$$C(k+1,i) = \left[2b + \sum_{j=0}^{k-1} 2\binom{k}{j} b\Delta_{j+1}(i) \right] \Delta_k(i) + C(k,i+1) - C(k,i). \quad (56)$$

It contains $\Delta_1(i), \ldots, \Delta_k(i)$, but not $\Delta_{k+1}(i)$. Thus, we proved that (53) is true for all k.

By regarding (53) as a second-degree polynomial of $\Delta_k(i)$, it is easy to see that all $E_k(i)$ are permutation polynomials and $\Delta_k(i)$ has the same order as the term $C(k,i)$.

Before finding the orders of all $C(k,i)$'s, we need to derive a compact expression for $C(k,i)$.

From (51), $C(1,i) = 2b\Delta_0(i)$. We can use (56) to inductively compute $C(k,i)$. For example

$$\begin{aligned}
C(2,i) &= 2b\Delta_1^2(i) + 4b\Delta_1(i), & (57)\\
C(3,i) &= 6b\Delta_2(i) + 6b\Delta_1(i)\Delta_2(i) + 6b\Delta_2^2(i) \\
&\;\;\vdots & (58)
\end{aligned}$$

It can be observed that

$$C(k,i) = 2kb\Delta_{k-1}(i) + \sum_{m=1}^{k-1} \sum_{n=1}^{k-1} C_{m,n}^{(k)} b\Delta_m(i)\Delta_n(i) \quad (59)$$

where $C^{(k)}$ forms a $(k-1) \times (k-1)$ matrix. If we define $\Delta^{(k)}(i) = [\Delta_1(i), \Delta_2(i), \ldots, \Delta_{k-1}(i)]$, (59) can be written in a matrix form

$$C(k,i) = 2kb\Delta_{k-1}(i) + \Delta^{(k)} C^{(k)} (\Delta^{(k)})^T. \quad (60)$$

Inductively, we can show that $C_{m,n}^{(k)} = 0$ when $m + n < k$ and for other m and n, if we define $\binom{x}{y} = 0$ and if $x < y$

$$C_{m,n}^{(k)} = \binom{k}{m} \binom{m}{k-n}. \quad (61)$$

It can be verified that $C^{(2)} = [2]$ and $C^{(3)} = \begin{bmatrix} 0 & 3 \\ 3 & 6 \end{bmatrix}$ satisfy (59). Applying (59) in (56)

$$\begin{aligned}
&C(k+1,i) \\
&= \left[2b + \sum_{j=0}^{k-1} 2\binom{k}{j} b\Delta_{j+1}(i) \right] \Delta_k(i) + 2kb\Delta_{k-1}(i+1) \\
&\quad + \sum_{m=1}^{k-1} \sum_{n=1}^{k-1} C_{m,n}^{(k)} b\Delta_m(i+1)\Delta_n(i+1) - 2kb\Delta_{k-1}(i) \\
&\quad - \sum_{m=1}^{k-1} \sum_{n=1}^{k-1} C_{m,n}^{(k)} b\Delta_m(i)\Delta_n(i) \\
&= 2(k+1)b\Delta_k(i) + \sum_{m=1}^{k-1} \sum_{n=1}^{k-1} bC_{m,n}^{(k)} \\
&\quad \times \Big(\Delta_{m+1}(i)\Delta_n(i) + \Delta_m(i)\Delta_{n+1}(i) \\
&\quad\quad + \Delta_{m+1}(i)\Delta_{n+1}(i) \Big) \\
&\quad + 2\sum_{j=0}^{k-1} \binom{k}{j} b\Delta_{j+1}(i)]\Delta_k(i) \\
&= 2(k+1)b\Delta_k(i) \\
&\quad + \Delta^{(k+1)}(D_1 + D_2 + D_3 + D_4 + D_5)(\Delta^{(k+1)})^T \\
&= 2(k+1)b\Delta_k(i) + \Delta^{(k)} C^{(k+1)} (\Delta^{(k)})^T
\end{aligned} \quad (62)$$

$$\begin{cases} C_{m,n}^{(k+1)} = 0, & m+n < k+1 \\ C_{k,1}^{(k+1)} = C_{k-1,1}^{(k)} + 1 & \\ C_{1,k}^{(k+1)} = C_{1,k-1}^{(k)} + 1 & \\ C_{k,n}^{(k+1)} = C_{k-1,n-1}^{(k)} + C_{k-1,n}^{(k)} + \binom{k}{n-1}, & 1 < n < k \\ C_{m,k}^{(k+1)} = C_{m-1,k-1}^{(k)} + C_{m,k-1}^{(k)} + \binom{k}{m-1}, & 1 < m < k \\ C_{k,k}^{(k+1)} = C_{k-1,k-1}^{(k)} + 2k & \\ C_{m,n}^{(k+1)} = C_{m-1,n}^{(k)} + C_{m,n-1}^{(k)} C_{m-1,n-1}^{(k)}, & \text{otherwise.} \end{cases} \tag{64}$$

where D_h's are $k \times k$ matrices for $h = 1, \dots, 5$. Define $\boldsymbol{g}^{(k)}$ to be a $1 \times k$ vector such that

$$\boldsymbol{g}^{(k)} = \left[\binom{k}{0}, \binom{k}{1}, \dots, \binom{k}{k-1} \right].$$

Then

$$\begin{aligned} D_1 &= \begin{pmatrix} \mathbf{0}_{1\times(k-1)} & 0 \\ C^{(k)} & \mathbf{0}_{(k-1)\times 1} \end{pmatrix} \\ D_2 &= \begin{pmatrix} \mathbf{0}_{(k-1)\times 1} & C^{(k)} \\ 0 & \mathbf{0}_{1\times(k-1)} \end{pmatrix} \\ D_3 &= \begin{pmatrix} 0 & \mathbf{0}_{1\times(k-1)} \\ \mathbf{0}_{(k-1)\times 1} & C^{(k)} \end{pmatrix} \\ D_4 &= \begin{pmatrix} \mathbf{0}_{k\times(k-1)} & \boldsymbol{g}^T \end{pmatrix}, \\ D_5 &= \begin{pmatrix} \mathbf{0}_{(k-1)\times k} \\ \boldsymbol{g} \end{pmatrix}. \end{aligned} \tag{63}$$

More precisely, see (64) at the top of the page. It is straightforward to check that (61) satisfies (64). So we proved that $C(k,i)$ satisfies (59).

Now we can find the order of all $C(k,i)$'s. Note that at this step of the induction, the orders of $\Delta_0(i), \Delta_1(i), \dots, \Delta_{k-1}(i)$ are known and satisfy (47) and (48).

We claim that the order of $C(k,i)$ is decided by the first term of (59), $2kb\Delta_{k-1}(i)$. From the previous step in the induction, we know

$$o_{\Delta_{k-1}} = (k-1)o_b + (k-1)o_2 + \sum_{n=1}^{k-1} o_n + o_s.$$

Then the first term of (59) has an order $ko_b + ko_2 + \sum_{n=1}^{k} o_n + o_s$. It is enough to show that all other terms have larger order. Consider the nth column of $C^{(k)}$, the first nonzero element is $C_{k-n,n}^{(k)}$. For $m > k-n$

$$\frac{C_{m,n}^{(k)}}{C_{k-n,n}^{(k)}} = \frac{n!}{(k-m)!(n-k+m)!} = \binom{n}{k-m}$$

is an integer. This implies that $C_{m,n}^{(k)}$ is a multiple of $C_{k-n,n}^{(k)}$ and $o_{C_{m,n}^{(k)}} \geq o_{C_{k-n,n}^{(k)}}$. Also, since $m < k$ and $k-n < k$, we know the order of Δ_m and Δ_{k-n} by the previous steps of the induction and $o_{\Delta_m} \gg o_{\Delta_{k-n}}$ for $m > k-n$. Combining these two results, we can see that in (59) the order of term $C_{m,n}^{(k)} b\Delta_m(i)\Delta_n(i)$ is strictly larger than the order of $C_{k-n,n}^{(k)} b\Delta_{k-n}(i)\Delta_n(i)$ for $m > k-n$. Then, in order to find the order of $C(k,i)$, we only need to consider the first term in (59) and terms in the form $C_{k-n,n}^{(k)} b\Delta_{k-n}(i)\Delta_n(i)$ for $n = 1, \dots, k-1$. By (48), the first term has an order

$$ko_b + ko_2 + \sum_{n=1}^{k} o_n + o_s.$$

The term in the form of $C_{k-n,n}^{(k)} b\Delta_{k-n}(i)\Delta_n(i)$ has an order

$$(k+1)o_b + ko_2 + \sum_{n=1}^{k} o_n + 2o_s$$

which is strictly larger than the order of the first term. Thus, the order of the first term in (59) is the order of $\Delta_k(i)$, which is

$$o_{\Delta_k(i)} = ko_b + ko_2 + \sum_{n=1}^{k} o_n + o_s, \qquad \forall\, i. \tag{65}$$

This concludes our proof.

Acknowledgment

The authors would like to thank Prof. Michael P. Fitz from the University of California at Los Angles for his active involvement in the research covered in this paper. They would also like to thank the anonymous reviewers for their helpful suggestions.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. Int. Conf. Communications*, Geneve, Switzerland, May 1993, pp. 1064–1070.

[2] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Proc. Int. Conf. Communications*, Seattle, WA, Jun. 1995.

[3] H. R. Sadjadpour, N. J. A. Sloane, M. Salehi, and G. Nebe, "Interleaver design for turbo codes," *IEEE J. Select. Areas Commun.*, vol. 19, no. 4, pp. 831–837, May 2001.

[4] C. Fragouli and R. D. Wesel, "Semi-random interleaver design criteria," in *Proc. GLOBECOM'99*, 1999, pp. 2352–2356.

[5] F. Daneshgaran and M. Mondin, "Design of interleavers for turbo codes: Iterative interleaver growth algorithms of polynomial complexity," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1845–1859, Sep. 1999.

[6] O. Y. Takeshita and D. J. Costello, Jr, "New deterministic interleaver designs for turbo codes," *IEEE Trans. Inf. Theory*, vol. 46, no. 6, pp. 1988–2006, Sep. 2000.

[7] *Recommendations for Space Data System Standards. Telemetry Channel Coding*, Consultative Committee for Space Data Systems Std. CCSDS 101.0-B-4. Blue Book, May 1999.

[8] L. Perez, J. Seghers, and D. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1698–1709, Nov. 1996.

[9] S. Crozier and P. Guinand, "Distance upper bounds and true minimum distance results for turbo-codes designed with DRP interleavers," in *Proc. 3rd Int. Symp. Turbo Codes*, Brest, France, Sep. 2003, pp. 169–172.

[10] C. J. Corrada-Bravo and I. Rubio, "Deterministic interleavers for turbo codes with random-like performance and simple implementation," in *Proc. 3rd Int. Symp. Turbo Codes*, Brest, France, Sep. 2003.

[11] R. L. Rivest, "Permutation polynomials modulo 2^w," *Finite Fields Their Applic.*, vol. 7, pp. 287–292, 2001.

[12] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*. Oxford, U.K.: Clarendon, 1975.

[13] W. W. Adams and L. J. Goldstein, *Introduction to Number Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1976.

[14] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 409–429, Mar. 1996.

[15] R. Garello, P. Pierleoni, and S. Benedetto, "Computing the free distance of turbo codes and serially concatenated codes with interleavers: Algorithm and applications," *IEEE J. Select. Areas Commun.*, vol. 19, pp. 800–812, May 2001.

[16] H. H. Ma and J. K. Wolf, "On tail biting convolutional codes," *IEEE Trans. Commun.*, vol. COM-34, no. 2, pp. 104–111, Feb. 1986.

[17] P. Stahl, J. B. Anderson, and R. Johannesson, "A note on tailbiting codes and their feedback encoders," *IEEE Trans. Inf. Theory*, vol. 48, no. 2, pp. 529–534, Feb. 2002.