A Novel Method for Obtaining the Structure of RTZ Inputs and their Corresponding Parity-Check Sequences: A First Step Towards Interleaver Design

Bohulu Kwame Ackah and Chenggao Han Graduate School of Informatics and Engineering,
The University of Electro-Communications,
1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182–8585, Japan
Email: {bohulu, han.ic}@uec.ac.jp

Abstract

Knowledge of the distance spectrum, as well as the structure of the message inputs that make up the distance spectrum for a specific recursive systematic convolutional (RSC) code is vital to the design of Turbo Code interleavers. Even though the distance spectrum of an RSC code can be obtained by calculating its transfer function, it does not provide any information about the structure of the message inputs.

In this paper,we present a novel low-complexity method for determining the distance spectrum of any RSC code that has the added benefit of revealing the structure of the Return-To-Zero (RTZ) inputs that make up the distance spectrum as well as their corresponding parity-check sequences. We then go a step further and present a method for deriving a general polynomial representation for both RTZ inputs and parity-check sequences with a Hamming weight of up to 4 for any RSC code.

Combining these two methods, we list the partial distance spectrum for selected RSC codes up to a cut-off weight $d_{\max}$ and compare the simulation results to the bounds obtained via our novel method and the transfer function method.

I. Introduction

The Turbo Code (TC) [1] is among the forward-error correcting codes that come very close to satisfying the Shannon limit for AWGN channels. It was introduced by Claude Berrou in 1993 and steadily gained fame and has been used in many applications and industry standards. It has been adopted as the standard channel code for the LTE standard, IEEE 802.16 WiMAX (worldwide interoperability for microwave access) and DVB-RCS2 (2nd generation digital video broadcasting - return channel via satellite) standards [7].

The simplest and most common construction of a TC is the parallel concatenation of two recursive systematic convolutional (RSC) codes (of the same kind) via an interleaver. One of the many reasons why the TC excels as a channel code is its ability to map low-weight parity-check sequences in one RSC code to high-weight parity-check sequences in the other, which in turn generates TCs with a large minimum distance value.

For this reason, interleaver design for TCs has been a hot topic for many years and generally, they are grouped into random and deterministic interleavers. Random interleavers determine their order of permutation in a pseudo-random manner. TCs using random interleavers usually have good error-correcting capabilities but imposes huge memory constraints for many practical applications due to the use of interleaver tables. A notable example of a random interleaver is the S-random interleaver.

On the flip side, deterministic interleavers generate their order of permutation via algorithms and, as such, can be generated on the fly, killing the need for permutation tables. Popular deterministic interleavers include quadratic permutation polynomial (QPP) interleaver [5], almost regular permutation (ARP) interleaver and dithered relative prime (DRP) interleaver. A protograph based interleaver design for punctured turbo codes is also introduced in [7] Deterministic interleavers also make it possible to perform parallel decoding once the interleaver meets certain requirements. Despite all these benefits, it is a well-known fact that in terms of TC error-correcting performance, random interleavers always outperform deterministic interleavers, especially for long frame sizes.

The design of good deterministic interleavers requires the complete knowledge of all input patterns that generate low-weight codewords in the component codes. We refer to these inputs as Return-to-Zero (RTZ) inputs [6]. Missing even one of these patterns can result in deterministic interleavers that generate TCs with sub-par error correction performance.

The transfer function of an RSC code is an interleaver design tool that provides information about the different weights in a code as well as their corresponding multiplicities (distance spectrum). However, it provides no information with regards to the pattern of the RTZ inputs. As an added downside, the complexity of calculating the transfer function for a given RSC code increases with the number of states. To the best of our knowledge, there exists no interleaver design tool that provides knowledge of the distance spectrum and the RTZ input patterns. Because of this many of the interleavers that have been designed based on assumptions which completely ignore certain important RTZ inputs. In [5] for example, the interleaver design method does not take into account the existence of RTZ inputs of weight 3, especially for the 5/7 RSC code where this RTZ input is very dominant with a high multiplicity.

In this paper,we present a novel method that can be used to find the distance spectrum of an RSC code as well as the pattern of the RTZ inputs and their corresponding parity-check sequences. The complexity of our method is independent of the number of states of the RSC code and its ability to reveal the pattern of RTZ inputs of the RSC code makes it an excellent tool for use in interleaver design.

In order to validate our method, we we generate a partial distance spectrum for specific RSC codes and compare it to the lower bound obtained via the transfer function method. We also compare the bounds obtained using our novel method to simulation results. In both cases, it is observed that the values begin to converge as E_b/N_0 increases.

The remainder of the research paper is organised as follows. In Section II, we briefly introduce the notations and assumptions that will be used in the research paper. Then,we briefly review the RSC codes in Section III . Moving on to Section IV, we briefly describe how the distance spectrum of an RSC Code is obtained using the transfer function, followed by the presentation of our novel method. Simulation results are presented in Section V and conclude in Section VI.

II. Preliminaries

Binary vectors are represented by bold font and $\dot{\boldsymbol{v}}$ represent an infinite repetition of the vector $\boldsymbol{v}$ whiles $(\boldsymbol{v})_j$ represents the repetition of vector $\boldsymbol{v}$ j times

$\boldsymbol{\phi}$ represents a primitive element in the extended Galois field GF(2^τ) and the vector representation for all elements in GF(2^τ) are written as $\boldsymbol{\phi}_i,\ 0 \leq i \leq 2^\tau$.

The subscript i represents the decimal value of the binary vector, which means $\boldsymbol{\phi}_0$ represents the all-zero vector. All addition and multiplication operations are done in GF(2^τ).

Also the operation (e mod M, f mod M) is represented by (e, f) mod M, where (e, f) are integer pairs.

III. Recursive Systematic Convolutional Codes:Review

A rate $R = k/n$ recursive systematic convolutional (RSC) code receives an input of k bits and outputs n bits at each time instance. The output bits are generated using feedback and feedforward connections of a shift register determined by generator functions. The generator function for the RSC code may be written in polynomial notation as $\left[1 \ \frac{f(x)}{g(x)}\right]$, where $f(x)$ and $g(x)$ represent the feedforward and feedback connections of the shift register respectively. Also, 1 represents the systematic (input) bits that make up the output.

We only focus on the parity part of the RSC code specified by the generator function $\frac{f(x)}{g(x)}$

Then, the impulse response caused by the feedback connection of the RSC encoder is easily calculated as

$$\phi_g(x) = g^{-1}(x)$$

and its equivalent vector representation takes the form,

$$\dot{\boldsymbol{\phi}}_g$$

where $\boldsymbol{\phi}_g$ is known as the cycle of the RSC code and τ is the cycle length.

The impulse response of the RSC code is calculated as

$$\phi(x) = f(x)\phi_g(x)$$

By careful observation, we can choose two integer values h and r such that we can write the equivalent vector representation as

$$\boldsymbol{\phi} = \boldsymbol{\phi}_h \dot{\boldsymbol{\phi}}_r$$

The minimum distance ($d_{\min}$) of the RSC code determines its error-correcting capability. With the aid of the distance spectrum, it is possible to determine $d_{\min}$ as well as its multiplicity. The most common way to find the distance spectrum is via the transfer function of the RSC code. The transfer function enumerates all the paths that diverge from

and then return to the initial state [3], i.e. the RTZ inputs. In other words, the distance spectrum provides information about the number of codewords of weight d generated by an RTZ input of weight w. Combined with the knowledge that all RTZ inputs are divisible by $g(x)$ [6], we introduce a novel method for determining the pattern of the input bits which cause low-weight codewords of any RSC code in the next section. We refer to this method as the codeword pattern distance spectrum. This version of the distance spectrum is a better tool for interleaver design compared to the regular distance spectrum.

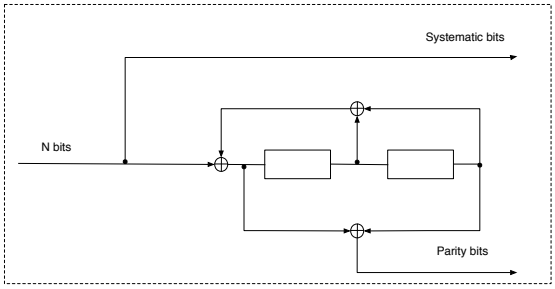

Fig. 1. $[\frac{1+x^2}{1+x+x^2}]$ RSC Encoder

Example 1. An RSC encoder is shown in Figure 1 with $k = 1$ and $n = 2$. Its parity generator function is given by $[\frac{1+x^2}{1+x+x^2}]$, which may be written as 5/7 in octal form, where 5 and 7 correspond to the numerator and denominator of the generator function, respectively. For this RSC code, the cycle is $\boldsymbol{\phi}_g = \boldsymbol{\phi}_6$ with a cycle length $\tau = 3$. While $\boldsymbol{\phi} = (1\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ \cdots)$, which may be written in terms of the elements of GF(8) as $\boldsymbol{\phi}_7\ \dot{\boldsymbol{\phi}}_3$ represents the impulse response.

Moving forward all other examples and discussions relating to RSC codes will be done using the 5/7 RSC code unless otherwise stated.

IV. Novel Method For Finding The Distance Spectrum

In this section, we present our novel method for obtaining what we have named the codeword pattern distance spectrum. Our novel method can be seen to be a combination of two different but related methods. The first method is quite simple and makes use of the fact that in the polynomial domain, RTZ inputs and their corresponding parity-check sequences share a common factor. The second method shows how to obtain this common

factor when the weight of the RTZ input or its parity-check sequence is fixed for a given RSC code. After explaining the inner working of our novel method, we used it to obtain the partial structured distance spectrum for the 5/7, 37/21 and 23/35 RSC codes.

Throughout this section, the symbols $\boldsymbol{b}$, $\boldsymbol{h}$ and $\boldsymbol{c}$ represent the input message, parity-check sequence and the codeword, respectively in vector form, while $b(x)$, $h(x)$ and $c(x)$ are their equivalent polynomial representations.

Before we go into the details of our novel method, we will briefly discuss how to obtain the distance spectrum of the RSC code using its transfer function.

A. Distance Spectrum via the Transfer Function of an RSC code

The distance spectrum of an RSC code gives information about codeword weights and the number of codewords present in the code for a given weight generated as a result of message inputs that begin from, exit and then return to the zero state of the trellis of that code. Such message inputs are known as Return-to-Zero (RTZ) inputs. The distance spectrum of the RSC code can be obtained from its transfer function, denoted by

$$T(Y, X) = \sum_{d=0}^{\infty} \sum_{w=0}^{\infty} a(d, w) Y^d X^w$$

where $a(d, w)$ is the number of codewords of weight d generated by an input message of weight w. Interested readers are referred to [3] where the transfer function for the 5/7 RSC code is derived. The complexity involved in deriving the transfer function increases as the number of states of the RSC code increases and other methods such as Mason's Rule [3] have to be used. Also to obtain the distance spectrum requires an extra division operation. In the case of interleaver design for turbo codes, this method for generating the distance spectrum is not particularly useful. This is because it reveals no extra details with respect to the structure of the RTZ inputs. Next, we present our novel method, whose complexity is independent of the number of states in the RSC code. As an added bonus, information regarding the structure of the RTZ inputs can be obtained using this method.

B. Low-weight Codewords

In polynomial notation, the parity-check sequence can be expressed as

$$h(x) = f(x) \cdot g^{-1}(x) \cdot b(x) \tag{1}$$

If we consider large frame sizes, the presence of $g^{-1}(x)$ means that within $h(x)$ is a particular sequence of bits that is repeated a large number of times. This results in a large parity weight with high probability, leading to a codeword with a high weight. The only time this is not the case is when

$$b(x) \bmod g(x) \equiv 0 \tag{2}$$

This results in a relatively low-weight parity bit sequence, which might produce a low-weight codeword. Any $b(x)$ that meets the condition in (2) can be written as

$$b(x) = a(x)g(x) \tag{3}$$

where $a(x)$ is a monic polynomial with the coefficient of the lowest term not equal to 0. By fixing $b(x)$ from (3) into (1), we have

$$\begin{aligned} h(x) &= f(x) \cdot g^{-1}(x) \cdot a(x)g(x) \\ &= a(x)f(x) \end{aligned} \tag{4}$$

From (3) and (4), we observe that $a(x)$ is a common factor in both equations and if we are able to solve for $a(x)$ via either of the equations, the remaining equation can be solved. To solve for $a(x)$ requires that in either equation, it should be the only unknown variable. At first glance, it might seem that $g(x)$ and $f(x)$ are the only known variables because they are dependent on the RSC code in question. However, if we remember that the weight of $h(x)$ and $b(x)$ is directly proportional to the number of terms it has, then we are on our way to obtain our second known variable. What is left is to determine the valid power values for the polynomial terms, depending on the weight of $h(x)$ or $b(x)$.

Revisiting (4), once $f(x)$ is given, our goal is to find $a(x)$ that results in a low-weight $h(x)$. The this end, we consider the roots of $f(x)$ denoted by $\beta_i,\ 0 \leq i < 2^{\text{order(f(x))}}$. Then it is obvious that $h(\beta_i) = 0$ for all i and we can reformulate our goal as to find weight-w polynomials ($h(x)$) which take all values of β_i as its roots. Similarly, we attempt to find weight-w polynomials ($b(x)$) whose roots are the same as $g(x)$

C. Finding Valid Values of $h(x)$ and $b(x)$ for a fixed Hamming weight and RSC code

We have established that the relationship between the roots of $f(x)$ and $h(x)$, as well as the relationship between the roots of $g(x)$ and $b(x)$. Depending on the characteristic

make-up of $f(x)$, we can easily determine the structure of $h(x)$ for its different weight values. The same logic applies to determining the structure of $b(x)$ for its different weight values from $g(x)$. We therefore, present a method for determining valid values of $h(x)$ and $b(x)$ for a given RSC code when the weight values $w_H(\boldsymbol{h}),\ w_H(\boldsymbol{b}) \leq 3$. We state here that there is no RTZ input of weight 1, because it goes against the definition of RTZ inputs. Consequently, there also cannot be a corresponding parity-check sequence of weight 1 and we can ignore those cases. The characteristic make-up of both $f(x)$ and $g(x)$ can be grouped into the four cases below.

1) Single primitive polynomial.
2) Prime but not a primitive polynomial.
3) Made up of repeated polynomial roots.

Because there is no difference between the general structure of $h(x)$ and $b(x)$ once the Hamming weight is fixed, we will focus on only $g(x)$ and $b(x)$ (4) in the discussion that follows.

1) Solution for $w_H(\boldsymbol{b}) = 2$: For this weight case, we can write $b(x)$ as $b(x) = 1 + x^a$ without any loss of generality. Thus we wish to find the possible values of a satisfying

$$\beta^{ia} + 1 = 0$$

for all $\beta^i,\ 1 \leq i \leq 2^m - 1$ in GF$(2^m),\ m = \text{order}(g(x))$

a) Case1: $g(x)$ is a single primitive polynomial: Since $g(x)$ is a prime polynomial, it has a primitive element β^i as its root, which means, β^i is also a root of $b(x) = 1 + x^a$. Setting $i = 1$ without loss of generality and substituting β into the equation, we have

$$\begin{aligned} 1 + \beta^a &= 0 \\ \beta^a &= 1 \end{aligned} \tag{5}$$

For any primitive polynomial that generates the extended field GF(2^m), $\beta^{2^m-1} = 1$, which means any valid of a should satisfy the condition below.

$$2^m - 1 \mid a$$

Example 2. $g(x) = 1 + x + x^2$.
$g(x)$ generates the field GF(2^2) where $\beta^3 = 1$. The valid values of a are $a = \{3, 6, 9, \cdots\}$. The corresponding values for $a(x)$ and $b(x)$ are shown in the table below for the first four valid values of a.

TABLE I

$g(x) = 1 + x + x^2$

$a(x)$	$b(x)$
$1 + x$	$1 + x^3$
$1 + x + x^3 + x^4$	$1 + x^6$
$1 + x + x^3 + x^4 + x^6 + x^7$	$1 + x^9$
$1 + x + x^3 + x^4 + x^6 + x^7 + x^9 + x^{10}$	$1 + x^{12}$

Example 3. $g(x) = 1 + x + x^4$

$g(x)$ can be used to generate the extended field GF(2^4). In this field, $\beta^{15} = 1$. The valid values of a are $a = \{15, 30, 45, \cdots\}$. The corresponding values for $a(x)$ and $b(x)$ are shown in the table below for the first two valid values of a.

TABLE II

23/35 RSC Code, $f(x) = 1 + x + x^4$

$a(x)$	$b(x)$
$1+x^2+x^3+x^5+x^7+x^8+x^{11}$	$1 + x^{15}$
$1 + x^2 + x^3 + x^5 + x^7 + x^8 +$ $x^{11} + x^{15} + x^{16} + x^{17} + x^{18} +$ $x^{20} + x^{22} + x^{23} + x^{26}$	$1 + x^{30}$

b) Case2: $g(x)$ is prime polynomial but not primitive

: Similar to the case for primitive polynomials, there is a value $j < 2^m - 1$ such that

$$\beta^j = 1, \ j \mid 2^m - 1$$

. Therefore, any valid value of a should satisfy the condition below.

$$j \mid a$$

Example 4. $g(x) = 1 + x + x^2 + x^3 + x^4$

$g(x)$ can be used used to generate GF(2^4) and in the field it generates, $\beta^5 = 1$. Therefore, the valid values of a are $a = \{5, 10, 15, \cdots\}$. The corresponding values for $b(x)$ and $a(x)$ are shown in the table below for the first four valid values of a.

TABLE III
$f(x) = 1 + x + x^2 + x^3 + x^4$

$a(x)$	$b(x)$
$1 + x$	$1 + x^5$
$1 + x + x^5 + x^6$	$1 + x^{10}$
$1 + x + x^5 + x^6 + x^{10} + x^{11}$	$1 + x^{15}$
$1 + x + x^5 + x^6 + x^{10} + x^{11} + x^{15} + x^{16}$	$1 + x^{20}$

c) Case3: $r(x)$ is made up of repeated polynomial roots.

: We may write $r(x)$ as

$$r(x) = \prod_{k=1}^{K} r_k(x)$$

The kth polynomial is prime and has a value d_k s.t. $\beta^{j_k} = 1$. Because each value of j_k is unique, valid values of a should satisfy the condition below.

$$\bigcap_{k=1}^{K} \{j_k \mid a\}$$

For the special case where $g(x)$ has equal repeated roots, the above condition simplifies to

$$kj \mid a$$

Example 5. $g(x) = 1 + x^2$

$g(x)$ can be written as

$$f(x) = (1 + x)^2, \ k = 2$$

$1 + x$ is prime in $GF(2)$ and $\beta^1 = 1$. Since $k = 2$, we have $\beta^k = \beta^2 = 1$. $h(x) = 1 + x^a$ and valid values of $a = \{2, 4, 6, \cdots\}$. The corresponding values for $a(x)$ and $b(x)$ are shown in the table below for the first four valid values of a

Example 6. $g(x) = 1 + x^4$

$g(x)$ is made up of equal repeated polynomial roots and can be written as

$$g(x) = (1 + x)^4, \ k = 4$$

. $1 + x$ is prime in $GF(2)$ and $\beta^1 = 1$. Since $k = 4$, we have $\beta^k = \beta^4 = 1$. $b(x) = 1 + x^b$ and the valid values of $a = \{4, 8, 12, \cdots\}$. The corresponding values for $a(x), \ b(x)$ and $h(x)$ are shown in the table below for the first four valid values of a

TABLE IV

$f(x) = 1 + x^2$

$a(x)$	$b(x)$
1	$1 + x^2$
$1 + x^2$	$1 + x^4$
$1 + x^2 + x^4$	$1 + x^6$
$1 + x^2 + x^4 + x^6$	$1 + x^8$

TABLE V

$g(x) = 1 + x^4$

$a(x)$	$h(x)$
1	$1 + x^4$
$1 + x^4$	$1 + x^8$
$1 + x^4 + x^8$	$1 + x^{12}$
$1 + x^4 + x^8 + x^{12}$	$1 + x^{16}$

Example 7. $g(x) = 1 + x^2 + x^3 + x^4$

$g(x)$ can be written as

$$g(x) = (1 + x)(1 + x + x^3),\ K = 2$$

$1+x$ is prime in $GF(2^1)$ and $\beta^1 = 1$. $1+x+x^3$ is prime in $GF(2^3)$ and $\beta^7 = 1$. $b(x) = 1+x^b$ and consequently, the valid values of a that meet the condition

$$\bigcap_{k=1}^{K} \{j_k \ | a\}$$

are $a = \{7, 14, 21, \cdots\}$. The corresponding values for $a(x)$ and $b(x)$ are shown in the table below for the first four valid values of a

TABLE VI
$g(x) = 1 + x^2 + x^3 + x^4$

$a(x)$	$b(x)$
$1 + x^2 + x^3$	$1 + x^7$
$1 + x^2 + x^3 + x^7 + x^9 + x^{10}$	$1 + x^{14}$
$1 + x^2 + x^3 + x^7 + x^9 + x^{10} + x^{14} + x^{16} + x^{17}$	$1 + x^{21}$
$1 + x^2 + x^3 + x^7 + x^9 + x^{10} + x^{14} + x^{16} + x^{17} + x^{21} + x^{23} + x^{24}$	$1 + x^{28}$

2) Solution for $w_H(\boldsymbol{b}) = 3$: For this weight case,
$b(x) = 1 + x^u + x^v,\ v \neq u$ without loss of generality. Given $g(x)$, our task is to find valid $(u,\ v)$ pair values satisfying the condition

$$1 + \beta^u + \beta^v = 0$$

If there are no values for the pair (u, v), then $q(x)$ s.t. $w_H(\boldsymbol{q}) = 3$ does not exist for the given $g(x)$.

a) Case1: $r(x)$ is a single primitive polynomial

Because $r(x)$ is a prime polynomial, it has a primitive element β as its root, which means that β is also a root of $q(x) = 1 + x^u + x^v$. Substituting β into the equation, we have

$$\begin{aligned} 1 + \beta^u + \beta^v &= 0 \\ \beta^u + \beta^v &= 1 \end{aligned} \tag{6}$$

where $u \neq v$ By refering to the table of the extended field for GF(2^m), we can find the valid (e, f) pairs s.t. $\beta^e + \beta^f = 1,\ e \neq f$. If we represent the set of (e, f) pairs as $\boldsymbol{z} = \{(e_1, f_1), (e_2, f_2), \cdots\}$, then any valid value for u and v should satisfy the condition

$$(u, v) \equiv (e, f) \bmod 2^m - 1,\ (e, f) \in \boldsymbol{z}$$

since $\beta^{2^m - 1} = 1$.

Example 8. $g(x) = 1 + x + x^2$

The elements of GF(2^2) are shown in Table II and it is obvious that there is exactly 1 valid (e, f) pair s.t. $\beta^e + \beta^f = 1$ and that is the pair $(1, 2)$. This means that valid values of the

(u, v) pairs are any values s.t. $(u, v) \equiv (1, 2) \bmod 3$. The corresponding values for $a(x)$ and $b(x)$ are shown in the table below for the first four valid values of (u, v)

TABLE VII
Non-zero Elements of GF(2^2) generated by $g(x) = 1 + x + x^2$

power representation	actual value
$\beta^0 = \beta^3 = 1$	1
β	β
β^2	$1 + \beta$

TABLE VIII
$g(x) = 1 + x + x^2$

$a(x)$	$b(x)$
1	$1 + x + x^2$
$1 + x + x^2$	$1 + x^2 + x^4$
$1 + x + x^3$	$1 + x^4 + x^5$
$1 + x^2 + x^3$	$1 + x + x^5$

Example 9. $g(x) = 1 + x + x^4$
The elements of GF(2^4) are shown in Table IX and we can see that there are seven valid (e, f) pairs.

This means that the valid values of the (u, v) pairs are any values s.t. $(u, v) \equiv (e, f) \bmod 15, (e, f) \in \boldsymbol{z},\ \boldsymbol{z} = \{(1, 4),\ (2, 8),\ (3, 14),\ (5, 10),\ (6, 13),\ (7, 9),\ (11, 12))$. The corresponding values for $a(x)$ and $b(x)$ are shown in Table X below for the first four valid values of (u, v)

b) Case2: $r(x)$ is prime but not a primitive polynomial

A prime but not primitive polynomial generates an (extended) field with j elements, where $j < 2^m - 1$. Similar to the case where $r(x)$ is primitive, we confirm the existence of (e, f) pairs s.t. $\beta^e + \beta^f = 1$. If we represent the set of (e, f) pairs as $\boldsymbol{z} = \{(e_1, f_1), e_2, f_2), \cdots, (e_1, f_q)\}$, then any valid value for u and v should satisfy the condition

$$(u, v) \equiv (e, f) \bmod j,\ (e, f) \in \boldsymbol{z}$$

since $\beta^j = 1$

TABLE IX

Non-zero Elements of GF(2^4) generated by $g(x) = 1 + x + x^4$

power	polynomial
$\beta^0 = \beta^{15} = 1$	1
β	β
β^2	β^2
β^3	β^3
β^4	$\beta + 1$
β^5	$\beta^2 + \beta$
β^6	$\beta^3 + \beta^2$
β^7	$\beta^3 + \beta + 1$
β^8	$\beta^2 + 1$
β^9	$\beta^3 + \beta$
β^{10}	$\beta^2 + \beta + 1$
β^{11}	$\beta^3 + \beta^2 + \beta$
β^{12}	$\beta^3 + \beta^2 + \beta + 1$
β^{13}	$\beta^3 + \beta^2 + 1$
β^{14}	$\beta^3 + 1$

TABLE X

$f(x) = 1 + x + x^4$

$a(x)$	$b(x)$
1	$1 + x + x^4$
$1 + x + x^2 + x^3 + x^5$	$1 + x^7 + x^9$
$1 + x + x^2 + x^3 + x^5 + x^7 + x^8$	$1 + x^{11} + x^{12}$
$1 + x + x^4$	$1 + x^2 + x^8$

Example 10. $g(x) = 1 + x + x^2 + x^3 + x^4$

From the table of the extended field generated by $g(x)$ (Table XI), we see that there are no valid $(e,\ f)$ and as such $b(x)$ s.t. $w_H(\boldsymbol{b}) = 3$ is non-existent.

c) Case3: $g(x)$ is made up of repeated polynomial roots

We may write $g(x)$ as

$$r(x) = \prod_{k=1}^{K} r_k(x)$$

TABLE XI
Non-zero Elements of GF(2^4) generated by $f(x) = 1 + x + x^2 + x^3 + x^4$

power	polynomial
$\beta^0 = \beta^5 = \beta^{10} = \beta^{15} = 1$	1
$\beta = \beta^6 = \beta^{11}$	β
$\beta^2 = \beta^7 = \beta^{12}$	β^2
$\beta^3 = \beta^8 = \beta^{13}$	β^3
$\beta^4 = \beta^9 = \beta^{14}$	$\beta^3 + \beta^2 + \beta + 1$

For the kth polynomial, we refer to the (extended) field table and determine if there exists any $(e^{(k)},\ f^{(k)})$ pairs s.t. $\beta^{e^{(k)}} + \beta^{f^{(k)}} = 1$. If it exists for all K polynomials, then the valid values for u and v should satisfy the condition

$$\bigcap_{k=1}^{K}(u, v) = (ku', kv')$$
$$, \ (u, v) \equiv (e^{(k)}, f^{(k)}) \bmod 2^m - 1, \ (e^{(k)}, f^{(k)}) \in \boldsymbol{z}^{(k)}$$

A special case is when $g(x)$ has equal and repeated roots. The above condition simplifies to

$$(u, v) = (ku', kv')$$
$$, \ (u, v) \equiv (e, f) \bmod 2^m - 1, \ (e, f) \in \boldsymbol{z}$$

.

Example 11. $g(x) = 1 + x^2$
$g(x)$ can be written as $(1 + x)^2$. $(1 + x)$ is a primitive polynomial for GF(2). The elements in GF(2) are 1 and β. In this field, there are no valid (e, f) pair values; therefore, $h(x)$ s.t. $w_H(\boldsymbol{h}) = 3$ does not exist.

Example 12. $g(x) = 1 + x^4$.
After factorisation, we have $g(x) = (1 + x)^4$.(1+x) is a primitive polynomial that generates the field GF(2) and in this field, there are also no valid $(e,\ f)$ and as such $b(x)$ s.t. $w_H(\boldsymbol{b}) = 3$ is also non-existent.

Example 13. $g(x) = 1 + x^2 + x^3 + x^4$.
Upon factorising, we have $g(x) = (1 + x)(1 + x + x^3)$. Table XII shows the elements of

GF(2^3) generated by $(1+x+x^3)$ and we can confirm that there are three valid $(e,\ f)$ pairs. However, since there are no valid $(e,\ f)$ pairs in GF(2), it also means that there cannot be any valid $(u,\ v)$ pairs and $b(x)$ s.t. $w_H(\boldsymbol{b}) = 3$ does not exist.

TABLE XII
Non-zero Elements of GF(2^3) generated by $1 + x + x^3$

power	polynomial
$\beta^0 = \beta^7 = 1$	1
β	β
β^2	β^2
β^3	$\beta + 1$
β^4	$\beta^2 + \beta$
β^5	$\beta^2 + \beta + 1$
β^6	$\beta^2 + 1$

D. Union Bound and the Codeword Pattern Distance Spectrum

Having determined how to find valid values of $b(x)$ and $h(x)$ for Hamming weights ≤ 3, we are now in a position to generate the codeword pattern distance spectrum for a given RSC code. We take a union bound like approach towards the generation of the codeword pattern distance spectrum. The approach is outlined below.

1) Beginning with (3), we find all values of $b(x),\ w_H(\boldsymbol{b}) = 2$ that have the same roots as $g(x)$ and divide $g(x)$ by each valid polynomial to obtain the corresponding $a(x)$.
2) Then using (4), we multiply each $a(x)$ by $f(x)$ to obtain the corresponding value of $h(x)$. It is worth noting that $w_H(\boldsymbol{h})$ may be $\geq w_H(\boldsymbol{b})$.
3) Since we are interested in only the low weight codewords, we ignore any $b(x)$ s.t. $w_H(\boldsymbol{b}) + w_H(\boldsymbol{h}) \leq d_{\max}$.
4) Next we set the weight value of $b(x)$ to $w_H(\boldsymbol{b}) = 3$, and repeat steps 1 and 2 while ignoring $b(x)$ that meet the condition in step 3.
5) To obtain a complete codeword pattern distance spectrum, we do a reverse operation, i.e. we focus on (3) and find all values of $g(x),\ w_H(\boldsymbol{h}) = 2$ that have the same roots as $f(x)$ and divide $f(x)$ by each valid polynomial to obtain the corresponding $a(x)$.
6) Then using (3), we repeat steps 2 through 4, being careful to avoid repitition.

7) Finally we arrange all valid values of $b(x)$ and $h(x)$ in ascending value of codeword weight,$w_H(\boldsymbol{c}) = w_H(\boldsymbol{b}) + w_H(\boldsymbol{h})$.

The codeword pattern distance spectrum for the 5/7, 37/21 and 23/35 RSC codes are shown in Tables XIII, XIV and XV respectively.

TABLE XIII

Partial Structured Distance Spectrum for the 5/7 RSC code, $d_{\max} = 8$

$a(x)$	$b(x)$	$h(x)$
1	$1+x+x^2$	$1+x^2$
$1+x^2$	$1+x+x^3+x^4$	$1+x^4$
$1+x$	$1+x^3$	$1+x+x^2+x^3$
$1+x^2+x^4$	$1+x+x^3+x^5+x^6$	$1+x^6$
$1+x^2+x^3$	$1+x+x^5$	$1+x^3+x^4+x^5$
$1+x+x^2$	$1+x^2+x^4$	$1+x+x^3+x^4$
$1+x+x^3$	$1+x^4+x^5$	$1+x+x^2+x^5$
$1+x^2+x^4+x^6$	$1+x+x^3+x^5+x^7+x^8$	$1+x^8$
$1+x+x^3+x^4$	$1+x^6$	$1+x+x^2+x^4+x^5+x^6$
$1+x+x^3+x^5$	$1+x^4+x^6+x^7$	$1+x+x^2+x^7$
$1+x+x^2+x^4$	$1+x^2+x^5+x^6$	$1+x+x^3+x^6$
$1+x+x^2+x^3$	$1+x^2+x^3+x^5$	$1+x+x^4+x^5$
$1+x^2+x^3+x^5$	$1+x+x^6+x^7$	$1+x^3+x^4+x^7$
$1+x^2+x^3+x^4$	$1+x+x^4+x^6$	$1+x^3+x^5+x^6$
$1+x^2+x^4+x^5$	$1+x+x^3+x^7$	$1+x^5+x^6+x^7$

TABLE XIV

Partial Structured Distance Spectrum for the 37/21 RSC code,$d_{\max} = 9$

$a(x)$	$b(x)$	$h(x)$
$1+x$	$1+x+x^4+x^5$	$1+x^5$
1	$1+x^4$	$1+x+x^2+x^3+x^4$
$1+x+x^5+x^6$	$1+x+x^4+x^6+x^9+x^{10}$	$1+x^{10}$
$1+x+x^4+x^5$	$1+x+x^8+x^9$	$1+x^4+x^5+x^9$
$1+x^2$	$1+x^2+x^4+x^6$	$1+x+x^5+x^6$
$1+x+x^5$	$1+x+x^4+x^9$	$1+x^6+x^7+x^8+x^9$
$1+x+x^4$	$1+x+x^5+x^8$	$1+x^4+x^6+x^7+x^8$
$1+x^2+x^4$	$1+x^2+x^6+x^8$	$1+x+x^4+x^7+x^8$
$1+x^3+x^4$	$1+x^3+x^7+x^8$	$1+x+x^2+x^4+x^8$
$1+x^4+x^5$	$1+x^5+x^8+x^9$	$1+x+x^2+x^3+x^9$

TABLE XV

Partial Structured Distance Spectrum for the 23/35 RSC code,$d_{\max} = 10$

$a(x)$	$b(x)$	$h(x)$
$1+x^2+x^3$	$1+x^7$	$1+x+x^2+x^6+x^7$
1	$1+x^2+x^3+x^4$	$1+x+x^4$
$1+x+x^2+x^3+x^5$	$1+x^3+x^4+x^8+x^9$	$1+x^7+x^9$
$1+x+x^2+x^3+x^5+x^7+x^8$	$1+x+x^3+x^4+x^7+x^{12}$	$1+x^{11}+x^{12}$
$1+x^2+x^3+x^7+x^9+x^{10}$	$1+x^{14}$	$1+x+x^2+x^6+x^8+x^9+x^{13}+x^{14}$

We use the codeword pattern distance spectrum to calculate the bit-error bounds for each RSC and compare them to the bit-error bounds obtained via the distance spectrum as well as simulation results. We use the probability of bit-error in doing this and a more general formula for calculating P_b is shown below [4]:

$$P_b \leq \frac{1}{k} \sum_{d=d_{\text{free}}}^{\infty} w(d) Q\left(\sqrt{\frac{2dE_c}{N_0}}\right) \tag{7}$$

where $w(d) = \sum_{i=1}^{\infty} i\ a(d,i)$ and $a(d,i)$ is the number of codewords of weight d generated by an input message of weight i. If we set a limit on the maximum value of the codeword weight $d_{\max}$ we can rewrite (7) as

$$P_b \leq \frac{1}{k} \sum_{d=d_{\text{free}}}^{d_{\max}} w(d) Q\left(\sqrt{\frac{2dE_c}{N_0}}\right) \tag{8}$$

From the simulation results, we observed $d_{\max} = d_{\min} + 3$ is a sufficient value for obtaining the BER bounds.

V. Results

In this section, we compare the bounds obtained via our novel method to bounds obtained using the transfer function method as well as the simulation results for three RSC codes. For each RSC code and a frame size of $N = 64$, the codeword is BPSK modulated and transmitted over the AWGN channel. At the receiver end, the Viterbi algorithm is used to decode before a decision is made on the decoded sequence.

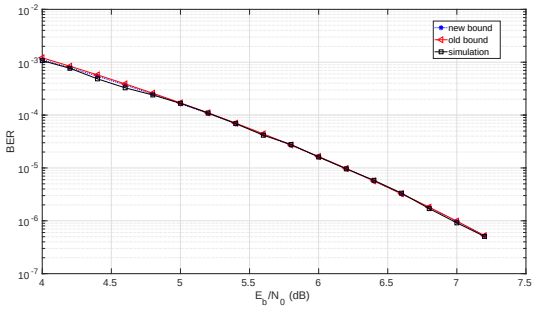

Fig. 2. Old Bound vs New Bound vs Simulation for 5/7 RSC Code

We observe that for both Fig. 2 and Fig. 3, there is some difference between the new (novel method) bound and the old (transfer function) bound, but they tend to converge as E_b/N_0 increases. This suggests that codewords generated considering $b(x),\ w_H(\boldsymbol{b}) > 3$ as well as codewords which have a parity-check sequence $h(x),\ w_H(\boldsymbol{h}) > 3$ do not have much effect on the BER of the code as E_b/N_0 increases.

However, in Fig. 4, it is apparent that codewords generated by higher weight RTZs as well as those with parity-check sequences with weights $w_H(\boldsymbol{h}) > 3$ still dominate at higher E_b/N_0 values and need to be considered. As can be observed from the graph, there is a very distinct gap between the new bound and the old bound. Moreover, the bounds do not converge as E_b/N_0 increases. However, the old bounds and simulation results converge as the E_b/N_0 value increases.

VI. Conclusion

In this In this paper, we presented a method for listing the codeword pattern distance spectrum for selected RSC codes up to a cut-off weight $d_{\max}$ by focusing on codewords

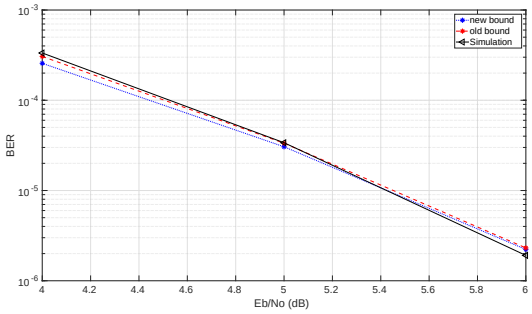

Fig. 3. Old Bound vs New Bound vs Simulation for 37/21 RSC Code

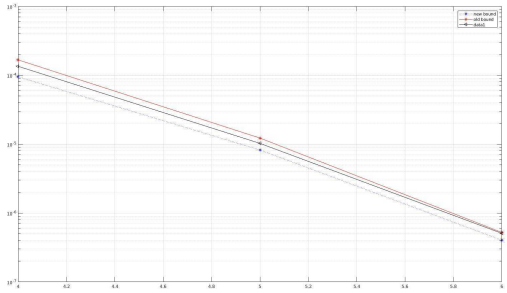

Fig. 4. Old Bound vs New Bound vs Simulation for 23/35 RSC Code

generated by the RTZ inputs of weight $w_H(\boldsymbol{b}) \leq 3$ as well as codewords with parity-check sequences $w_H(\boldsymbol{h}) \leq 3$. Compared to the transfer function method, it has low complexity and provides extra information with regard to the structure of the RTZ inputs as well as the parity-check sequences, which makes it very useful for interleaver design. We compared the bounds obtained using our novel method with the bounds obtained via the transfer function as well as the simulation results for three RSC codes. The results suggest that considering codewords generated by RTZ inputs of weight $w_H(\boldsymbol{b}) > 3$ as well as codewords with parity-check sequences $w_H(\boldsymbol{h}) > 3$ will improve the accuracy of the bounds obtained via our method.

References

[1] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," Proceedings of ICC '93 - IEEE International Conference on Communications, Geneva, Switzerland, 1993, pp. 1064-1070 vol.2, doi: 10.1109/ICC.1993.397441.

[2] John G. Proakis, Masoud Salehi. "Digital Communications", Fifth Edition,Chapter 8, McGraw-Hill.

[3] Todd K. Moon. "Error Correcting Codes",Chapter 12, John Wiley & Sons.

[4] Alain Glavieux, "Channel Coding in Communication Networks: From Theory to Turbocodes", Chapter 3, John Wiley & Son.

[5] Jing Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," in IEEE Transactions on Information Theory, vol. 51, no. 1, pp. 101-119, Jan. 2005, doi: 10.1109/TIT.2004.839478.

[6] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan and M. Jezequel, "Designing good permutations for turbo codes: towards a single model," 2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577), Paris, France, 2004, pp. 341-345, doi: 10.1109/ICC.2004.1312507.

[7] R. Garzón-Bohórquez, C. Abdel Nour and C. Douillard, "Protograph-Based Interleavers for Punctured Turbo Codes," in IEEE Transactions on Communications, vol. 66, no. 5, pp. 1833-1844, May 2018, doi: 10.1109/TCOMM.2017.2783971.