Computing the Free Distance of Turbo Codes and Serially Concatenated Codes with Interleavers: Algorithms and Applications

Kwame Ackah Bohulu

2017/11/21

1 Abstract

この論文ではインタリーバで並列連結された符号と直列された符号の最低距離 (d_free) を計算するのに使える新しいアルゴリズムを紹介する。d_{free} の値が分かれば、エラーフローの推定が分析的にできる。このアルゴリズムは、制限されたサブコードという新しい考え方に基づき、情報系列の重みを拘束されずに大きいインタリーバに対する大きい距離の計算が可能です。

2 Introduction

[1] で紹介されたターボ符号は、基本のバイナリ畳み込み符号をインタリーバで並列連結して作られて作られている。紹介された時から、色んな分野でスタンダードとして使われている。[5] で最初に紹介された直列された畳み込み符号 (SCCC) は、特に低いビット誤り率とフレーム誤り率の場合、ターボ符号よりいい性能を持つ。

受信側で、ターボ符号と SCCC 符号の復号化は BCJR アルゴリズムを繰り返す。ところが、高い SNR の場合、両方の符号の性能が悪くなる可能性がある。高い SNR の場合、バイナリ符号の性能は符号の最低距離 (d_{free}) と多重度 (multiplicity) に影響を与える。

[7] で明らかになったことは、あるインタリーバを使用する畳み込み符号は、大きいフレームサイズでも、小さい d_{free} の符号語を作る可能性があるということがわかった。そうなると、ビット誤り率性能のグラフには、エラーフローが出てくる。応用に 10^{-8} 10^{-10} ビット誤り率性能が必要なら、この動作は望ましくない。

なので、小さいビット誤り率の場合、連結された符号の性能を推定するのが大事です。d_{free} が分かれば、符号の性能が分析的に性能が確認でき、エラーフローもわかるようになる。多重度 N_{free} と情報ビット多重度 W_{free} 線形バイナリ符号 $C(n,k)$ を仮定すると、高い E_b/N_0 場合、以下の式が出てくる。

$$FER \simeq \frac{1}{2} N_{free} erfc\Big(\sqrt{d_{free}\frac{k}{n}\frac{E_b}{N_0}}\Big) \tag{1}$$

$$BER \simeq \frac{1}{2} \frac{w_{free}}{k} erfc\Big(\sqrt{d_{free}\frac{k}{n}\frac{E_b}{N_0}}\Big) \tag{2}$$

例 **1** ：N=8920 のレート 7/8 のターボ符号を考える。この符号は ($d_{free}/N_{free}/w_{free}$) = (2/4/8)。図 1 では、式 (1) で計算されたエラーフローとシミュレーションの結果を比べた。明らかに、ターボ符号の誤り率性能はエラーフローで制限されている。

また、入力重み 2 エラーイベントに対する値、d_2,N_2,w_2 を計算する。($d_2/N_2/w_2$)=(3/2/4)。入力重み 2 エラーイベントは union bound に与える影響を以下の式で計算する。

$$UB(2) \simeq \frac{1}{2} \frac{w_{free}}{k} erfc\Big(\sqrt{d_{free}\frac{k}{n}\frac{E_b}{N_0}}\Big) + \frac{1}{2} \frac{w_2}{k} erfc\Big(\sqrt{d_2\frac{k}{n}\frac{E_b}{N_0}}\Big)$$

以上の例で d_{free} を計算するのに使える効率的なアルゴリズムの重要さがわかりました。そういったアルゴリズムがないため、d_{free} を計算するとき、入力重み 2 エラーイベントしか使わない。この方法を使うと d_{free} の上界を得られる。

実用的な場合、フレーメサイズ $N \simeq 1000$ を使ってもこの方法は満足できない、そして誤解を招く結果が出てくる。

3 SCC 符号の紹介

SCC 符号のシステム図は　で示される。要素符号器 (CE) は 2 つのレート 1/2RSC 符号で作られている。CE2 のレートはパンクチュアリングで 2/3 に設計された。長さ N のインタリーバも付いている。符号化方法は以下で説明される。

[1]. k-ビットの情報系列 $\mathbf{u} = (u_0, ..., u_k)$ を CE1 に入力し、長さ 2k の符号系列を作る。CE1 を状態 0 に戻すため、2v ビットを追加し、出力ビット $\mathbf{c_1} = (c_{1,0}, ..., c_{1,N-1})$ の長さは、N=2(k+v) になる。

[2]. $\mathbf{c_1}$ をインタリーバで並び替えて、$\mathbf{u_2} = \Pi(\mathbf{c_1}) = (u_{2,0}, ..., u_{2,N-1})$ を CE2 入力する。CE2 を状態 0 に戻すため、2v ビットを追加する。

[3]. 符号語 $\mathbf{c}$ の長さ n は、N+v が偶数の場合、n=(N+v)3/2 N+v が奇数の場合、n=(N+v-1)3/2+2

4 Past Work on the Subject

4.1 Weight-Two input sequence algorithm

[13] で提案された方法は、d_{free} を計算するとき、重み - 2 入力シーケンスに対する符号語重み ($d^{(2)}_{free}$) しか使わない

この方法、インターリーバがランダム、そしてインターリーバの長さが非常に大きい場合、であれば $d^{(2)}_{free} = d_{free}$ が可能です。

必要な計算量は少なくなりますが、得られる値は上限である。

このアルゴリズムは、以下の理由により満足できるものではない

[1]. ンタリーバサイズ $N \simeq 1000$ の場合かパンクチュアリングした場合、[13] の結果は適用されません。

[2]. 計算された上限 $d^{(2)}_{free}$ が実際の値 d_{free} と等しいかどうかを確認する方法はない。

[3]. 非古典的な符号化体系には適用されない。

[4]. この方法は、短期から中程度のインタリーバフレームサイズの場合、d_{free} の値を過大評価する傾向がある。

この方法は、インターリーバの最適化にするのに役に立つ。結局、実際の d_{free} を計算する必要がある。

4.2 Error Event Algorithm

d_{free} の実際値を計算するアルゴリズムは、[7] と [8] に示されている。

手順は以下で説明されている。

[1]. まず、d_{free} を特定の値 d^* と仮定します。要素符号器を選んで、集合 S(全ゼロ状態 0 $_{sum}$ から出る、重みが d^* 未満の符号語を生成するすべての情報系列) を考慮する

[2]. S の各情報系列に対して、ターボ符号器によって符号化する符号語の重みを計算する。

[3]. S の情報系列のシフトされたバージョンに対して、上記ステップを繰り返す。

アルゴリズムの複雑さは N と $dfree$ に大きく依存します。N が大きくなるにつれて、集合 S は大きくなり、必要なシフト数もおおきくなる。

5 Constrained Subcodes

この論文で提示されているアルゴリズムは、制約付きサブコードの最小距離の計算に基づいています。

終端された畳み込み符号のみを考慮された。しかし、提示されたすべての技術は、任意のタイプの要素符号に容易に一般化することができる。

5.1 Definitions

端 $e_i = \{\sigma_i, c_i, \sigma_i + 1\}$ から生成される時間 i $(0 \leq i \leq t-1)$ における符号トレリス部分は $\Gamma_i = \{e_i\}$

トレリス上の許容可能な端経路と符号語との間には 1 対 1 の対応がある。

時間 i$(0 \leq i \leq t-1)$ での制約は組 $v_i = \{i, e\}\ , e \in \Gamma_i$

時間 i においてエ端 e を通過する場合、符号語は v_i を満たす。

複数の制約: $v_i = \{i, e_a, e_b, ...\}\ , e_a, e_b, ... \in \Gamma_i$ 時間 i においてエ端 e_a か e_b を通過する場合、符号語は v_i を満たす。

制約セット: $V = \{v_i\}$

制約サブコード $C(V)$: 制約セットを満たす符号語。

5.2 Computing the Minimum Distance of a Constrained Subcode

$\mathbf{c} \in C(v)$ の最小距離をコードワードの最小ハミング重みとして定義する。

拘束されたサブコードの最小距離の計算は、畳み込み符号の自由距離を計算するために使用される一般的なビタビ様の技法にいくつかの修正を適用することによって行うことができる。

通常に端の重みは、そのバイナリラベルのハミング重みと一致するが、制約 $v_i = \{i, e_a, e_b, ...\}$ あれば、v_i にいない端の重みを非常に大きな値に設定しなければならない。

アルゴリズムの説明は以下で説明される。

[1] 状態 0 から始まって、重みを 0 にする。

[2] 時間 i にする状態の重みを時間 i-1 に対する端と状態の重みで計算する。

[3] 時間 i=t のとき、端の重みが制約されたサブコードの重みになる。

5.3 Input Bits and Constrained Subcodes

以下では、入力ビットによって引き起こされる制約を考慮する。

レート-1/2 畳み込み符号のトレリス区間を考える。各状態を離れる 2 つのエッジがある。

結果として、時間 i における入力ビットに対して行われた仮説は、複数の制約を引き起こす。