A Method for Determining the Distance Pro£le of Turbo Codes

Marco Breiling and Johannes Huber *
Lehrstuhl für Nachrichtentechnik II
Universität Erlangen-Nürnberg
E-Mail: breiling@nt.e-technik.uni-erlangen.de
WWW: http://www.lnt.de/∼dcg

Abstract — **This contribution proposes a new algorithm for determining the low distance terms in the distance pro£le of Turbo codes by analyzing their interleaver. The method is applied to several interleavers and the results are displayed and discussed.**

I. Introduction

In 1993, Berrou et al. presented Turbo codes as a new class of very power ef£cient binary channel codes [1]. A characteristic of Turbo codes is the *error ¤oor*. This term represents the fact that after a steep descent of the Bit Error Rate (BER) curve at very low Signal-to-Noise Ratios (SNR), the BER curve ¤attens out more or less abruptly and then descends with a shallow slope. The position of the error ¤oor depends on the Turbo code structure, i.e. the rate, the particular interleaver and the convolutional codes employed. The reason for the error ¤oor are terms at comparatively low distances in the code's distance pro£le. When setting up a Turbo coding scheme, one has to choose an interleaver, which largely determines the Turbo code's performance. It is therefore important to assess the performance of the interleaver considered in order to avoid bad choices. This paper describes a new method, which can be used to determine the low distance terms in the distance pro£le of a Turbo code by analyzing the interleaver, when the encoder structure is given.

Following the introduction, we £rst present our notation and system model in Section II, which is followed by a description of the proposed algorithm in Section III. This method is used to analyze several Turbo code interleavers, of which the results are presented in Section IV. A summary of this paper is £nally given in Section V.

II. System model and terminology

In this Section, we present the model of the Turbo encoder used throughout this paper. As can be seen from Figure 1, the encoder consists of three parallel branches, which are connected to the encoder input. Each branch generates one part of the codeword $\mathbf{c} = (\mathbf{u}; \mathbf{c}^{(1)}; \mathbf{c}^{(2)})$. The upmost branch provides the codeword's systematic part, which is identical to the *information word* $\mathbf{u} = (u_1; ..; u_K)$ consisting of the K binary encoder input symbols generating one codeword. In the following discourse, the middle and the lower branches are referred to as the £rst and the second *component*, respectively. These branches generate the *component codewords* $\mathbf{c}^{(1)}$ and $\mathbf{c}^{(2)}$ forming the parity part of the codeword $\mathbf{c}$ by passing the information symbols through scramblers. Some authors prefer the term component codes for the scramblers.

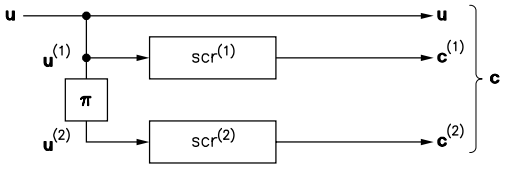

Fig. 1: System model of the Turbo encoder

The following terminology is used throughout the paper. A superscript index $\bullet^{(1)}$ or $\bullet^{(2)}$ denotes the component a variable $\bullet$ is associated with. The input of the £rst component scrambler $\text{scr}^{(1)}$ is the sequence $\mathbf{u}^{(1)} = \mathbf{u}$ of information symbols in their original order. For the second component, these information symbols are permuted by an interleaver $\boldsymbol{\pi} = (\pi_1; ..; \pi_K)$ to form the second component information word $\mathbf{u}^{(2)}$, which is then fed into the second component scrambler $\text{scr}^{(2)}$. The permutation follows the rule that $u^{(2)}_{\pi_i} = u^{(1)}_i$ for $i = 1..K$. In the paper, we assume without loss of generality that there is no puncturing of the component codewords $\mathbf{c}^{(1)}$ and $\mathbf{c}^{(2)}$, i.e. the code rate is $1/3$. Furthermore, we assume that both component scramblers are identical and that they are binary shift-registers comprising a feed-forward and a feed-back branch as examplarily shown in Figure 2. These assumptions are made for the sake of simplicity, but an extension of the method presented is possible also for the case of puncturing, i.e. higher rates, and two differing component scramblers. Moreover, we will always assume that both component scramblers are *terminated* (i.e. in the zero-state) after the input of $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$, respectively. This can be accomplished in the Turbo encoder by the method described in [2].

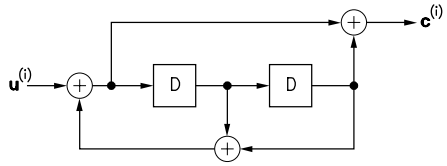

Fig. 2: Example of a component scrambler with feed-back polynomial 7 and feed-forward polynomial 5

In order to explain our interleaver analysis method, we need to introduce the following term. An *error pattern* is a scrambler input sequence that at £rst drives the associated scrambler out of the zero-state, then leads it through

* The authors would like to thank the Fraunhofer Gesellschaft – Institut für Integrierte Schaltungen, Erlangen, for supporting this work.

arbitrary many non-zero states, and £nally brings it back to the zero-state. The name error pattern is attributed to the following fact. Since Turbo codes are linear, for a performance evaluation we can take as a reference the transmission of the all-zero information word, and determining the distance pro£le is identical to determining the distribution of all codeword weights. Then every *erroneously* decoded information word is composed of at least one *error* pattern and otherwise "0"s. An error pattern is associated with an *error event*, which denotes a path deviating from and returning to the zero-path in the scrambler's trellis. For scramblers employed in a Turbo encoder (i.e. the parity symbol output of *recursive* systematic convolutional component codes), there is in fact feed-back, i.e. the polynomial describing the feedback branch is $\neq 1$. Hence for these scramblers, all error patterns have at least weight 2, since the error pattern has to start with a "1" to let the associated path deviate from the zero-path, and it has to end with a "1" to make the path return to the zero-state. Some examples of error patterns for the scrambler of Figure 2 are shown in Table I.

TABLE I
Three error patterns of low weight (= input weight) generating low output weight for the scrambler of Figure 2.

error pattern	weight	generated scrambler output weight
111	3	2
1001	2	4
10101	3	4

A scrambler is terminated after K input symbols, if and only if the scrambler input sequence comprises only error patterns and "0"s between them. Since we assume a termination of both component scramblers, both $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ are composed only of error patterns and "0"s. Figure 3 shows an example, where the £rst component information word $\mathbf{u}^{(1)}$ contains two error patterns "1001" of Table I. The transpositions performed by the interleaver $\boldsymbol{\pi}$ are indicated by arrows in the Figure. We recognize that through the permutation the same error pattern appears twice in the second component information word $\mathbf{u}^{(2)}$. From Table I we see that each pattern "1001" has weight 2 and generates a scrambler output of weight 4. Since the weight $w(\mathbf{c})$ of a Turbo codeword is the sum of the weights of its three constituent parts $w(\mathbf{c}) = w(\mathbf{u}) + w(\mathbf{c}^{(1)}) + w(\mathbf{c}^{(2)})$, we can easily calculate that the generated codeword has weight $2{\cdot}2{+}2{\cdot}4{+}2{\cdot}4 = 20$.

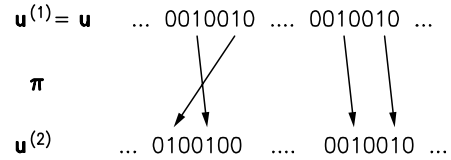

Fig. 3: 1st component information word containing two error patterns permuted to a similar 2nd component information word

III. Determining the distance profile

In this Section, we propose a new algorithm for examining a given interleaver in order to determine the distance pro£le of the Turbo code employing this interleaver. In a £rst step, we will describe a simple analysis method, which has already been proposed by several authors [3],[4]. On the basis of this simple method, we are going to develop a more sophisticated algorithm, which is £rst presented in an example before it is described in detail.

The simple and the new algorithm have in common that they proceed stepwise through the interleaver. In each step, only those terms of the distance pro£le, which are associated with a *subset* of all codewords are determined. More speci£cally, the set $\mathcal{U} \setminus \{\mathbf{0}\}$ of all information words except the zero-word is subdivided into K disjoint subsets $\mathcal{U}_1; ..; \mathcal{U}_K$ with $\bigcup_{l=1}^{K} \mathcal{U}_l = \mathcal{U} \setminus \{\mathbf{0}\}$. Thereby, also all the associated codewords are subdivided into K disjoint subsets $\mathcal{C}_1; ..; \mathcal{C}_K$. The set $\mathcal{U}_l$ contains all information words $\mathbf{u}$, for which the following condition holds: $u_l = 1$ and $u_i = 0\ \forall i \in \{l+1; ..; K\}$. If we write down the elements of $\mathbf{u}$ in an ascending order, this means that the *right-most* "1" in $\mathbf{u}$ is located in position l. In step $l = 1..K$ of the algorithm, we consider only the information words belonging to set $\mathcal{U}_l$, and we determine the terms of the distance pro£le, which are due to the associated codewords in $\mathcal{C}_l$. After K steps, we have considered all information words/codewords except the zero-word in our analysis, such that we obtain the complete distance pro£le by combining the terms found during steps $1..K$.

Since it would be necessary to consider all the $2^K - 1$ non-zero codewords to determine the complete distance pro£le, we will restrict our analysis to the low distance terms in order to make the complexity manageable. We will therefore de£ne a value δ_{rel} representing the maximum *relevant* distance that we will take into account. We will hence only determine the terms of the distance pro£le for distances $\leq \delta_{\mathrm{rel}}$ during the analysis. The information words/codewords that cause these terms will be called the *relevant* information words/codewords, respectively. *Relevant Information Word* will be abbreviated by *RIW* in the following.

At the start of both algorithms, a list $\mathbf{E}$ of all *Relevant Error Patterns* (abbreviated by *REP*) is set up by searching for the associated error events in the scrambler trellis. We de£ne relevant error patterns as the set containing all constituent error patterns of all relevant information words and no other error patterns. A necessary condition for any REP is that its weight *plus* the weight of the associated scrambler output *plus* the minimum scrambler output weight for *any* (£nite-length) error pattern must be $\leq \delta_{\mathrm{rel}}$. If we set up the list $\mathbf{E}$ according to this condition, then we can be assured that all REPs are contained in $\mathbf{E}$. Since the above condition is not suf£cient, $\mathbf{E}$ does also contain additional irrelevant error patterns. This fact does however not disturb the execution of the algorithm.

A. Simple method

For this simple method, which has been proposed by [3] and [4] in a similar way, the number of information words to take into account is restricted even further than described above. We will consider only those information words, which are relevant for the given δ_{rel} value and which contain only a *single* error pattern. The distance pro£le obtained by

this simple method is exact only for the lowest distances, because the information words generating codewords of these weights do in fact contain only a single error pattern. However, the terms for higher distances resulting from this simple method are smaller than in the exact pro£le, since many of the RIWs are ignored. Considering Figure 3 makes this clearer. Only the information words, where *either* the left *or* the right of the two error patterns is present and where "0"s are present instead of the other one, are taken into account in the distance pro£le obtained through this simple method. However the information word containing *both* error patterns is never considered in this analysis method and its participation in the associated distance pro£le term is neglected.

At the start of the simple algorithm, a list $\mathbf{E}$ is set up as described above. Then steps $l = 1..K$ are executed:

In step l we make the following *hypothesis* on the information word $\mathbf{u}$. We assume that $\mathbf{u} \in \mathcal{U}_l$ contains only "0"s apart from a single error pattern from $\mathbf{E}$, which is placed in $\mathbf{u}$, such that its £nal (i.e. right-most) "1" is located in u_l. We know thus $w(\mathbf{u})$ and $w(\mathbf{c}^{(1)})$, which are the weights associated with the error pattern and its output. We then permute $\mathbf{u}^{(1)} = \mathbf{u}$ to $\mathbf{u}^{(2)}$ and check, whether it is made up of only REPs and "0"s. In this case, we can determine $w(\mathbf{c}^{(2)})$ as the sum of the output weights of the constituent error patterns of $\mathbf{u}^{(2)}$. We have hence found that our hypothesis information word is associated with a codeword of weight $w(\mathbf{c}) = w(\mathbf{u}) + w(\mathbf{c}^{(1)}) + w(\mathbf{c}^{(2)})$. If $w(\mathbf{c}) \leq \delta_{\rm rel}$, we can take this codeword into account by incrementing a counter for the associated term in the distance pro£le. Otherwise – or if $\mathbf{u}^{(2)}$ is not only made up of REPs – we can ignore this information word, since the associated codeword has a weight $> \delta_{\rm rel}$.

By making this hypothesis in step l for all REPs in $\mathbf{E}$, we £nally obtain the terms in the distance pro£le, which are due to all RIWs $\in \mathcal{U}_l$ containing only a single error pattern.

After step K, all terms are combined to form the complete distance pro£le.

B. Example of the new algorithm

In this Subsection, we will £rst describe a straightforward extension of the simple method of Subsection III.A, which takes into account cases as that of Figure 3. Another extension, being based on backtracking and forming the core of our new algorithm, is £rst shown in an example, before it is described in detail in Subsection III.C.

The case of Figure 3 can be taken into account by the following extension. Whenever a hypothesis information word has proven to be a RIW, it is stored in a set $\mathcal{U}_{\rm rel}$ of already found RIWs. When a new RIW $\mathbf{u}_1$ has been included in $\mathcal{U}_{\rm rel}$, we simply try to combine this with any previously found RIW $\mathbf{u}_2 \in \mathcal{U}_{\rm rel}$. The combination $\mathbf{u}_3 = \mathbf{u}_1 \oplus \mathbf{u}_2$ (elementwise addition in GF(2)) is a RIW, too, and must be inserted into $\mathcal{U}_{\rm rel}$, if the following two conditions are met. None of the constituent error patterns of the new RIW $\mathbf{u}_1$ may overlap with any of the constituent error patterns of $\mathbf{u}_2$ – neither in the 1st ($\mathbf{u}_1$ and $\mathbf{u}_2$) nor in the associated 2nd component information words ($\mathbf{u}_1^{(2)}$ and $\mathbf{u}_2^{(2)}$), and the sum of the two codeword weights associated with $\mathbf{u}_1$ and $\mathbf{u}_2$ must be $w(\mathbf{c}_1) + w(\mathbf{c}_2) \leq \delta_{\rm rel}$.

This algorithm extension does still not take into account all RIWs. Consider the case of Figure 4. The two error patterns in $\mathbf{u}$ are woven with those in $\mathbf{u}^{(2)}$ and – unlike Figure 3 – they do not represent a combination of two separate RIWs. We call this combination a *woven* error pattern. The basic idea for the new algorithm is to £nd a woven error pattern by recursively following its "threads" – the arrows in the Figures representing transpositions of the "1"s by the interleaver. We shall now examplarily present the work of the proposed algorithm for such a case.

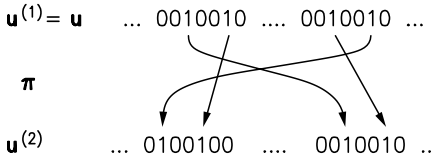

Fig. 4: Two error patterns in the information word constituting a woven error pattern

Step l of the new algorithm starts like step l of Subsection III.A. We make the hypothesis that $\mathbf{u}$ contains only a single error pattern from $\mathbf{E}$, which is located with its right-most "1" in u_l. For our example, which is displayed in Figure 5a), this error pattern is "1001". Those elements of the information word, which are part of an error pattern, are shaded in the Figure. We then £nd the associated $\mathbf{u}^{(2)}$ for this hypothesis by permuting $\mathbf{u}$. The "1"s in u_{l-3} and u_l correspond to "1"s in $u_i^{(2)}$ and $u_j^{(2)}$, respectively. We then try to *modify* the hypothesis $\mathbf{u}^{(2)}$, such that every "1" in the original $\mathbf{u}^{(2)}$ becomes a part of a REP in the modi£ed $\mathbf{u}^{(2)}$. In the example of Figure 5b), we replace "0"s in $u_{j+3}^{(2)}$ and $u_{i-3}^{(2)}$ by "1"s, as a result of which the modi£ed hypothesis $\mathbf{u}^{(2)}$ is now made up of two error patterns "1001" and "0"s between them. Figure 5c) shows that we then £nd the associated modi£ed hypothesis $\mathbf{u}$ by de-interleaving $\mathbf{u}^{(2)}$. The "1"s in $u_{j+3}^{(2)}$ and $u_{i-3}^{(2)}$ correspond to "1"s in u_{m+3} and u_m. By examining the modi£ed $\mathbf{u}$, we realize that these new "1"s establish an error pattern "1001". The modi£ed $\mathbf{u}$ contains hence two error patterns and "0"s between them, and the associated $\mathbf{u}^{(2)}$ is made up of only error patterns and "0"s, too. We have hence found a woven error pattern as shown in Figure 4. We can compute that the associated codeword weight is 20 (cf. Table I). If $20 \leq \delta_{\rm rel}$, then the modi£ed $\mathbf{u}$ is a RIW that has to be included in $\mathcal{U}_{\rm rel}$ and taken into consideration in the distance pro£le.

In the above example, we have seen that the new algorithm £nds woven error patterns by £rst following "threads" (i.e. pursuing "1"s through the interleaver), then placing new error patterns containing the existing "1"s in $\mathbf{u}^{(2)}$. Thereby new "1"s are produced in $\mathbf{u}^{(2)}$, which have to be pursued back through the de-interleaver etc. When a new error pattern is to be placed at the end of a "thread" (cf. Figure 5b)), all possible error patterns have to be considered in order to £nd all woven error patterns. This makes the pro-

posed algorithm a classical backtracking algorithm, which is presented in more detail in the next Subsection.

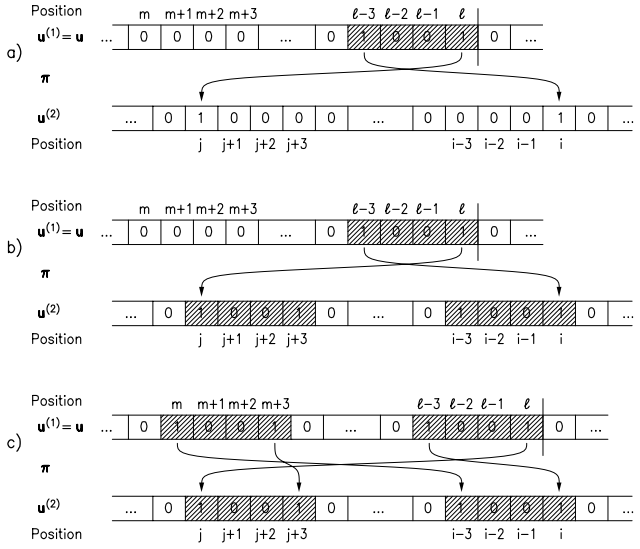

Fig. 5: Example of the new algorithm

C. Description of the backtracking algorithm

This Subsection describes the work of the new backtracking algorithm, which was examplarily shown in Subsection III.B and replaces the simple method of Subsection III.A. The other extension, which was introduced at the beginning of Subsection III.B, is used to £nd combinations with all other RIWs from $\mathcal{U}_{\mathrm{rel}}$, whenever a new woven error pattern has been found by the backtracking algorithm.

At the start of the interleaver analysis, the list $\mathbf{E}$ of REPs is set up as described above. Then, analysis steps $l = 1..K$ are executed.

For step l, let us de£ne the two procedures $\mathrm{backtrack}^{(1)}(\mathbf{u}^{(1)}(r), \mathbf{s}^{(1)}(r), \mathbf{s}^{(2)}(r))$ and $\mathrm{backtrack}^{(2)}(\mathbf{u}^{(2)}(r), \mathbf{s}^{(1)}(r), \mathbf{s}^{(2)}(r))$ calling each other recursively, where $\mathbf{u}^{(i)}(r)$ denotes the (modi£ed) hypothesis for $\mathbf{u}^{(i)}$ at recursion level r of the backtracking process. $\mathbf{s}^{(1)}(r)$ and $\mathbf{s}^{(2)}(r)$ are status vectors, where a component $s_i^{(1)}(r) = 1$ means that element $u_i^{(1)}$ belongs to an error pattern in $\mathbf{u}^{(1)}(r)$, otherwise $s_i^{(1)}(r) = 0$. The role of $\mathbf{s}^{(2)}(r)$ follows analoguously. These status vectors were represented in Figure 5 by shading positions.

$\mathrm{backtrack}^{(1)}(\mathbf{u}^{(1)}(r), \mathbf{s}^{(1)}(r), \mathbf{s}^{(2)}(r))$: Create all possible modi£ed hypotheses $\mathbf{u}^{(1)}(r+1)$ based on $\mathbf{u}^{(1)}(r)$ according to the following rules. The error patterns, which are already present in $\mathbf{u}^{(1)}(r)$ (i.e. the status vector elements $s_i^{(1)}(r) = 1$ for the associated positions i of $\mathbf{u}^{(1)}(r)$), must also be present in the same positions of $\mathbf{u}^{(1)}(r+1)$. Furthermore, new error patterns are placed in $\mathbf{u}^{(1)}(r+1)$, such that they do not overlap with the other error patterns in $\mathbf{u}^{(1)}(r+1)$, that every "1" in $\mathbf{u}^{(1)}(r)$ has a "1" in the corresponding position of $\mathbf{u}^{(1)}(r+1)$ and that at least one "1" of each error pattern in $\mathbf{u}^{(1)}(r+1)$ corresponds to a "1" in $\mathbf{u}^{(1)}(r)$. These rules ensure that $\mathbf{u}^{(1)}(r+1)$ contains the same error patterns and the same "1"s outside error patterns as $\mathbf{u}^{(1)}(r)$ plus "1"s belonging to additional error patterns. A further condition is that no "1" may be in position i of $\mathbf{u}^{(1)}(r+1)$, if $u_i^{(1)}(r) = 0$ and $s_{\pi_i}^{(2)}(r) = 1$. This ensures that a "0" in an existing error pattern (indicated by $u_i^{(1)}(r) = 0$ and $s_{\pi_i}^{(2)}(r) = 1$) cannot be set to "1" in the modi£ed hypothesis, because this would change the existing error pattern. Apart from the error patterns placed according to the above rules, $\mathbf{u}^{(1)}(r+1)$ contains only "0"s. Create all possible hypotheses (i.e. use all possible combinations of error patterns) based on the above rules. For each one of these hypotheses, update the status vector $\mathbf{s}^{(1)}(r+1)$ accordingly and call $\mathrm{backtrack}^{(2)}(\mathbf{u}^{(2)}(r+1), \mathbf{s}^{(1)}(r+1), \mathbf{s}^{(2)}(r+1))$, where the associated $\mathbf{u}^{(2)}(r+1)$ is obtained by interleaving $\mathbf{u}^{(1)}(r+1)$, and where $\mathbf{s}^{(2)}(r+1) = \mathbf{s}^{(2)}(r)$.

$\mathrm{backtrack}^{(2)}(\mathbf{u}^{(2)}(r), \mathbf{s}^{(1)}(r), \mathbf{s}^{(2)}(r))$: This procedure is analog to $\mathrm{backtrack}^{(1)}$ (simply exchange the two components) except for a further condition for the created hypotheses. The hypotheses $\mathbf{u}^{(2)}(r+1)$ may not contain a "1" in any element $u_{\pi_i}^{(2)}(r+1)$, for which $i > l$, i.e. where this "1" is right of position l of the associated $\mathbf{u}^{(1)}(r+1)$. This condition ensures that we restrict ourselves to hypotheses $\mathbf{u}^{(1)} \in \mathcal{U}_l$ in this analysis step l.

Having de£ned the recursion procedures, the start of the backtracking algorithm for step l is similar to the simple method from Subsection III.A. All possible hypotheses $\mathbf{u}^{(1)}(0)$ are set up, which contain only "0"s apart from a single error pattern from $\mathbf{E}$, whose £nal, i.e. right-most "1" is located in $u_l^{(1)}(0)$. The status vector $\mathbf{s}^{(1)}(0)$ is set to "1" for all positions belonging to this error pattern and "0" for all others. Then $\mathbf{u}^{(1)}(0)$ is permuted to the associated $\mathbf{u}^{(2)}(0)$ and the backtracking is started at recursion level $r = 0$ by invoking $\mathrm{backtrack}^{(2)}(\mathbf{u}^{(2)}(0), \mathbf{s}^{(1)}(0), \mathbf{s}^{(2)}(0))$, where all elements of $\mathbf{s}^{(2)}(0)$ are set to "0".

As described above, the procedures would call each other in£nitely. There are however two termination criteria, which are identical for $\mathrm{backtrack}^{(1)}$ and $\mathrm{backtrack}^{(2)}$. When one of these criteria is true, the procedure stops immediately and returns control to the calling procedure. When a procedure at recursion level $r+1$ returns to the calling procedure at level r, all variables of level r must be reset to the same state as before the call of level $r+1$; this is characteristic of recursive algorithms including backtracking algorithms. Only the counters for the distance pro£le may be modi£ed, if a RIW was found at level $r+1$. The two termination criteria are:

1. The sum of the weight of the hypothesis $\mathbf{u}^{(1)}(r)$ and the associated output weights of the two component scramblers surpasses the given δ_{rel} value for the error patterns, which have been placed in the two components so far. Then all codewords generated by this combination of error patterns (and possibly additional) have a weight $> \delta_{\mathrm{rel}}$ and are thus

not relevant.

2. A woven error pattern has been found. Then $\mathbf{u}^{(1)}(r)$ is a RIW, when $\mathrm{backtrack}^{(1)}$ is called (analog for $\mathrm{backtrack}^{(2)}$). In this case $\mathbf{u}^{(1)}(r)$ and $\mathbf{u}^{(2)}(r)$ contain only error patterns and "0"s between them, but no "1"s outside of error patterns. This case was depicted in Figure 5c). Then the RIW must be taken into account in the distance pro£le (by incrementing a counter), and the algorithm returns from this recursion level r.

Furthermore, the backtracking procedure returns of course to the calling procedure, when all possible hypotheses $\mathbf{u}^{(i)}(r+1)$ based on $\mathbf{u}^{(i)}(r)$ have been created and checked for at this recursion level r. When the lowest recursion level $r = 0$ returns, step l of the algorithm is £nished.

The algorithm has been implemented according to the above description with some minor modi£cations and optimizations as to improve the implementation and speed up the interleaver analysis. Reference [4] gives some ideas, how the number of error patterns to take into consideration in the backtracking procedures can be reduced, since not all error patterns $\in \mathbf{E}$ are relevant on a given recursion level r.

IV. Analysis results

In this Section, we present some of the results that have been obtained by the interleaver analysis algorithm. Both random and designed interleavers were analyzed for Turbo codes using memory $M = 3$ component scramblers with feed-back polynomial 13_8 (in octal) and feed-forward polynomial 17_8. In all cases, both component scramblers were supposed to be terminated after K input symbols. No puncturing was performed, and the code rate R is hence slightly smaller than $1/3$, since 6 of the symbols in the information word are used to terminate both component scramblers. In the analysis, the list $\mathbf{E}$ was restricted to contain only REPs of weight ≤ 8, unless stated otherwise. It is however highly unlikely that low weight codewords are generated by information words containing error patterns of weight > 8.

A. Random interleavers vs. Uniform Interleaver

The proposed algorithm was used in order to analyze the distance pro£le of Turbo codes employing random interleavers. An individual analysis was executed for 100 interleavers of length $K = 100$ and $K = 1000$, respectively. Then the distance pro£les were averaged over the 100 interleavers. Table II shows the results for interleaver length $K = 100$. Two kinds of coef£cients are displayed. The coef£cient A_δ denotes the number of codewords of weight δ, which is used in the Union Bound for upper bounding the Word Error Rate (WER) for Maximum Likelihood Decoding (MLD) [5]. In the Table, $A_{\delta,\mathrm{uni}}$ represents the value obtained from the Uniform Interleaver [5], i.e. by averaging over all $K!$ possible interleavers of length K. $\bar{A}_\delta$ denotes the value obtained by averaging over the 100 analyzed interleavers. $\bar{A}_{\delta,2}$ was obtained from the same 100 interleavers, but here only error patterns of weight 2 were taken into account (i.e. $\mathbf{E}$ was restricted to REPs of weight 2).

For upper bounding the Bit Error Rate (BER) for MLD, the coef£cients $D_\delta = \bar{w}_\delta \cdot A_\delta / K$ are used (cf. [5],[6]), where $\bar{w}_\delta$ denotes the average weight of the information word associated with codewords of weight δ. $D_{\delta,\mathrm{uni}}$ and $\bar{D}_\delta$ represent the terms for the uniform interleaver and the average over 100 analyzed interleavers, respectively. We see that the coef£cients for the (theoretical) uniform interleaver agree very well with the average over real random interleavers. We recognize that only a part of the distance pro£le is due to weight 2 error patterns for $K = 100$. As regards to Turbo code interleaver design, from these results we £nd that design methods are not suitable for short interleavers, if they only avoid error patterns of weight 2 generating low weight codewords.

TABLE II
Coef£cients A_δ and D_δ of the distance pro£le for $K = 100$

δ	$A_{\delta,\mathrm{uni}}$	$\bar{A}_\delta$	$\bar{A}_{\delta,2}$	$D_{\delta,\mathrm{uni}}$	$\bar{D}_\delta$	$\bar{D}_{\delta,2}$
8	0.002350	0.01	0	0.000094	0.0004	0
10	0.013766	0	0	0.000551	0	0
11	0.505850	0.51	0	0.015176	0.0153	0
12	0.082167	0.04	0	0.003289	0.0018	0
13	0.656365	0.63	0	0.019871	0.0191	0
14	2.019895	1.94	1.71	0.045885	0.0434	0.0342
15	2.138627	2.14	0	0.065297	0.0664	0
16	0.870876	0.9	0	0.035246	0.0368	0
17	2.977293	2.8	0	0.094823	0.0896	0
18	5.502618	5.16	2.98	0.158687	0.15	0.0596
19	6.765702	7.03	0	0.234684	0.2457	0

For interleaver length $K = 1000$, Figure 6 depicts the coef£cients $A_{\delta,\mathrm{uni}}$ for the uniform interleaver and the average $\bar{A}_\delta$ over 100 analyzed interleavers. For the important (i.e. large) terms of the distance pro£le, these two terms are almost identical. The Figure also displays the coef£cients $\bar{A}_{\delta,2}$ for considering weight 2 error patterns only; these coef£cients are greater than 0 only for $\delta = 14$ and $\delta = 18$, and here all three points are on top of each other in the Figure. We see that in contrast to $K = 100$, the important terms of the distance pro£le are almost entirely due to weight 2 error patterns, which can however be a part of a woven error pattern or a combination of error patterns (see Figures 4 and 3). This increasing importance of the weight 2 error patterns for the distance pro£le for growing interleaver length K was explained as "spectral thinning" in [6].

The spectral thinning effect has an impact on the distribution of the minimum distance $\delta_{\min}$ for random interleavers. This behaviour is shown in Table III, which displays the number of interleavers among the 100 analyzed interleavers, for which the associated Turbo code has the indicated $\delta_{\min}$ value. No interleaver was found with $\delta_{\min} = 9$ or $\delta_{\min} = 10$. For $K = 100$, all 100 interleavers had a $\delta_{\min} \leq 17$, and $\delta_{\min} \in \{11; 13; 14\}$ appeared with similar frequency. For $K = 1000$ however, almost all interleavers had $\delta_{\min} = 14$, which is due to weight 2 error patterns. All 100 interleavers had $\delta_{\min} \leq 18$. As the interleaver length K grows, distance pro£le terms remain only for those distances, which are due to weight 2 error patterns. Asymptotically for $K \to \infty$, only those minimum distances exist, that are due to these weight 2 error patterns, too (i.e. $\delta_{\min} \in \{14; 18; ...\}$ for this Turbo code). A large interleaver

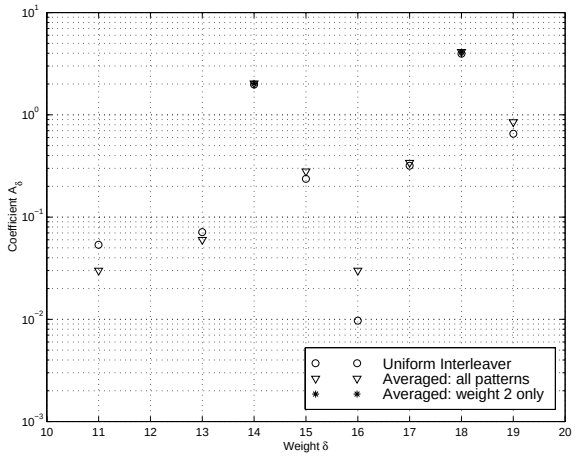

Fig. 6: Coef£cients A_δ of the distance pro£le for $K = 1000$

length has therefore several advantages: An interleaver gain for the BER (cf. [5]), an interleaver gain also for the WER, since low distance terms disappear, and a smaller variance of $\delta_{\min}$, such that most interleavers have the same $\delta_{\min}$.

B. Designed interleavers

Two interleavers have been analyzed, that were designed to increase $\delta_{\min}$ for the Turbo code described above. The £rst one is given in [7] and was designed for $K = 105$. The analysis result is displayed in Table IV (only distances $\delta \leq 25$ were considered). The distance pro£le provided for this interleaver by Hokfelt and Maseng in [7] differs from our results. The reason is that we assume a termination of both component scramblers, whereas Hokfelt and Maseng terminate only the £rst component scrambler. The second designed interleaver of [8] for $K = 1176$ was kindly provided by K. Andrews. To reduce the analysis complexity, we restricted the list $\mathbf{E}$ to REPs of weight ≤ 4. Because of spectral thinning for $K = 1176$, we can argue that the analysis is correct with high probability despite this restriction. In fact, only error patterns of weight 2 were involved in generating the only non-zero distance pro£le term for $\delta \leq 30$. This term is exclusively caused by the case of two error patterns "10000001" in $\mathbf{u}$ permuted to two identical patterns in $\mathbf{u}^{(2)}$ as similarly shown in Figure 4. All £ve woven error patterns found are of this type. We recognize that for both designed interleavers, $\delta_{\min}$ was clearly increased compared to random interleavers of similar length. When comparing with the Gilbert-Varshamov-Bound, we £nd for $R = 1/3$ a value of $\delta_{\min} = 55$ for information word length $K = 100$ and $\delta_{\min} = 521$ for $K = 1000$. A convolutional code of rate $1/3$ with similar decoding complexity per information bit (assuming 8 iterations in the Turbo decoder for the described Turbo code of $M = 3$) has 256 states and $\delta_{\min} = 18$.

V. CONCLUSION

We have proposed a method for analyzing an interleaver in order to obtain the distance pro£le of a Turbo code. We have used this algorithm for random interleavers and found that the variance of the minimum distance becomes less for larger interleaver lengths. The analysis showed that the Uniform Interleaver provides a good tool to describe the distance pro£le averaged over many random interleavers. Spectral thinning effects were observed. Two designed Turbo code interleavers were analyzed and discussed. Since the presented algorithm is capable of detecting low weight Turbo codewords, it could be used in an interleaver design algorithm in order to discard these codewords: if a Turbo codeword of too low a weight is found during an interleaver design, an element of the interleaver has to be changed to a value, for which there are no low weight codewords. An algorithm based on this principle has already been implemented and is currently being examined.

TABLE III
Number of analyzed interleavers with minimum distance $\delta_{\min}$

$\delta_{\min}$	8	11	12	13	14	15	16	17	18
$K = 100$	1	42	3	27	21	3	1	2	0
$K = 1000$	0	3	0	6	83	1	0	2	5

TABLE IV
Coef£cients A_d for designed interleavers of length $K = 105$ and $K = 1176$

δ	22	23	24	25	26	27	28	29	30
Hokfelt	4	7	17	52					
Andrews	0	0	0	0	0	0	5	0	0

VI. REFERENCES

[1] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo codes. *International Conference on Communications*, pages 1064–1070, 1993.

[2] O. Jörssen and H. Meyr. Terminating the trellis of turbocodes. *Electronics Letters*, 30(16):1285–1286, 1994.

[3] Patrick Robertson. Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes. *Global Conf. on Comm.*, pages 1298–1303, 1994.

[4] Fred Daneshgaran and Marina Mondin. An ef£cient algorithm for obtaining the distance spectrum of turbo codes. *International Symp. on Turbo Codes, Brest*, pages 251–254, 1997.

[5] Sergio Benedetto and Guido Montorsi. Unveiling turbo codes: Some results on parallel concatenated coding schemes. *IEEE Transactions on Information Theory*, 42(2):409–428, 1996.

[6] Lance Perez, Jan Seghers, and Daniel Costello. A distance spectrum interpretation of turbo codes. *IEEE Transactions on Information Theory*, 42(6):1698–1709, 1996.

[7] Johan Hokfelt and Torleiv Maseng. Methodical interleaver design for turbo codes. *International Symp. on Turbo Codes, Brest*, pages 212–215, 1997.

[8] Kenneth Andrews, Chris Heegard, and Dexter Kozen. Interleaver design methods for turbo codes. *Intern. Symposium on Inf. Th.*, page 420, 1998.