Interleaver structures for turbo codes with reduced storage memory requirement

Hokfelt, Johan; Edfors, Ove; Maseng, Torleiv

Published in:
Proc. IEEE Vehicular Technology Conference

DOI:
10.1109/VETECF.1999.801562

1999

Interleaver Structures for Turbo Codes with Reduced Storage Memory Requirement

Johan Hokfelt, Ove Edfors and Torleiv Maseng
Department of Applied Electronics, Lund University, Lund, Sweden

Abstract— **This paper describes two interleaver structures that reduce the memory requirement for the storage of interleaver rules. The first structure offer memory reductions of more than 50%, compared to storing the entire interleaver vector. The second structure is useful when a range of interleaver sizes are to be stored, offering memory reductions asymptotically approaching 50%. By combining the two structures, a total memory reduction of approximately 75% is achieved. This is obtained without any significant reduction of the error correcting performance of the codes.**

I. Introduction

A turbo code typically consists of two recursive encoders in parallel, separated by an interleaver [1] as shown in Fig. 1. The design parameters of a turbo code are primarily the generator polynomials of the constituent encoders, normally chosen to be identical, and the particular choice of interleaver mapping. The issue of choosing generator polynomials is discussed in for example [2], [3], [4], while for interleaver design strategies and interleaver structures, see for example [5], [6], [7], [8], [9].

In some applications, when many, and possibly large, interleavers are used, the implementation complexity to generate the interleaver rules is of importance. For example, in the standardization of UMTS, such considerations have achieved significant attention. There are basically two approaches that can be used to lower the implementation complexity of interleaver rules: either by using algebraic interleavers that can be generated in real time, or by imposing structures on the interleavers, which reduce the storage requirement of the interleaver rules.

This paper presents two interleaver structures that reduce the amount of memory required to store interleaver rules. The first structure is referred to as an *odd-even symmetric* structure, which reduces the memory requirement with slightly more than 50% compared to storing the entire interleaver vector. The second structure, *expanded* interleavers, is useful when a series of interleavers with sizes of the form

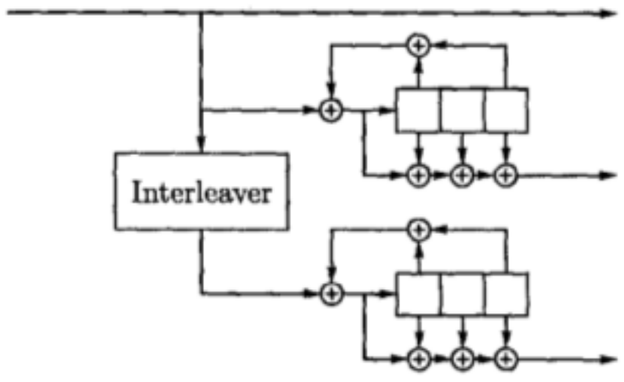


Fig. 1. Example of a turbo code encoder, using constituent encoders with generator polynomials $(17/15)_{oct}$.

$2^k x, k = 0, 1, 2, \ldots$, where x is the size of the smallest desired interleaver, are to be stored. When a series of interleavers are designed with the expansion criterion, it is sufficient to store the largest interleaver only, offering memory savings approaching 50%. Fortunately, the interleaver structures can be combined, with a total memory requirement reduction of approximately 75%, compared to storing all the interleaver vectors.

Naturally, every interleaver restriction reduces the design freedom, and hereby, possibly, deteriorates the performance of the interleavers. However, simulations show that interleavers with the structures presented in this paper perform essentially the same as unconstrained interleavers, if designed appropriately.

II. Interleaver structures

Let the interleaver rule be represented by a vector of N integers, π=[$\pi(1)$ $\pi(2)\ldots$ $\pi(N)$], where $\pi(i) = j$ indicates that input position i is interleaved to position j, and N is the size of the interleaver. The interleaver structures presented in this paper impose certain restrictions on the permissible choices of the mappings $\pi(i)$. First, we describe two different restrictions that individually offer implementation simplifications, and thereafter these restrictions are merged into a combined structure, offering total storage savings of as much as 75%.

A. *Odd-even symmetric interleaver structure*

Consider an interleaver rule that swaps pairs of positions, i.e. a symmetric interleaver. If all the pairs are known, the interleaver rule is known. Since there are only $N/2$ such pairs, if organized properly, the storage of these pairs requires less memory than storing an entire interleaver vector with N addresses. One possible organization strategy is to require every position in the first half of the input sequence to be swapped with a position in the second half. However, this restriction severely reduces the design freedom of the interleaver, notably deteriorating the error correcting performance of the code. There are however other sequence partitions that yield a simple organization of the swapping pairs, without degrading the interleaver performance. One such partition is to swap every odd position with an even position, and vice versa. This interleaver structure, denoted odd-even symmetric, is thus achieved with the following two restrictions:

1. $i \bmod 2 \neq \pi(i) \bmod 2, \forall i$ (odd to even),
2. $\pi(i) = j \Rightarrow \pi(j) = i$ (symmetry).

With these restrictions, it is sufficient to store the interleaver rules for all the odd positions, since by performing swaps, the even positioned bits are automatically interleaved.

Assume that only the odd positions in the interleaver vector are stored. All the stored addresses are then even integers, implying that the least significant bit (LSB) in the binary representation of each address is always zero. Thus, the LSB need not be stored, which offers additional memory savings if the interleaver rule is stored with custom made memory cells. This shift of the binary representation corresponds to dividing each number by 2, so that the stored vector consists of $N/2$ integers ranging from 1 to $N/2$. This vector will in the following be denoted $\tilde{\pi}$, and it is given by $\tilde{\pi}(i) = \pi(2i-1)/2, i \in \{1, 2, \ldots, N/2\}$.

As an example, the swapping pairs of an 8-bit odd-even symmetric interleaver are illustrated in Fig. 2. The shown interleaver vector is π=[6 3 2 7 8 1 4 5], and the reduced memory-requirement vector is $\tilde{\pi}$=[3 1 4 2].

The implementation of the interleaving rule of an odd-even symmetric interleaver is straightforward: elements at even positions are interleaved by storing them sequentially and reading them in the order specified by $\tilde{\pi}$; elements at odd positions are interleaved by storing them in the order specified by $\tilde{\pi}$ and reading them sequentially. As an example, we study the interleaving of the extrinsic outputs produced by the first constituent decoder. For illustrative purposes, it is suitable to partition the memory used to store the extrinsic information between the decoders into two logically separated memory areas, A and B. With these, odd extrinsic outputs on the form $2n-1$, $n \in \{1, 2, \ldots, N/2\}$ are stored at address $\tilde{\pi}(n)$ in memory A, while even outputs, $2n$, $n \in \{1, 2, \ldots, N/2\}$, are stored at address n in memory B. The second constituent decoder performs a similar action when reading its extrinsic inputs: odd inputs are read from memory B at address $\tilde{\pi}(n)$, and even inputs are read from memory A at address n. Such an interleaver implementation is illustrated in Fig. 3. The deinterleaver implementation is identical, due to the symmetric property.

Fig. 2. Example of an 8-bit odd-even symmetric interleaver. Each odd position in the input sequence is mapped to an even position, and vice versa. Further, if input i is mapped to position j, then input j is mapped to position i (symmetry).

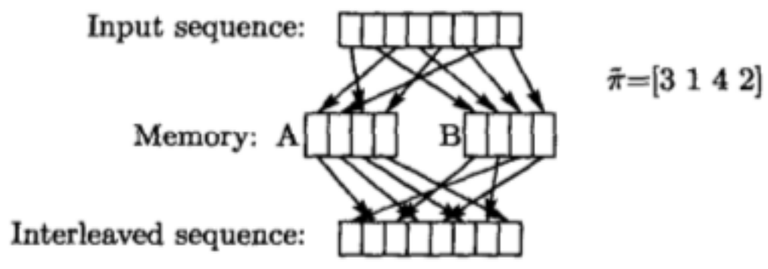


Fig. 3. Implementation example of an 8-bit odd-even symmetric interleaver. The interleaver rule is stored by the 4-element vector $\tilde{\pi}$=[3 1 4 2].

B. *Expanded interleaver structure*

The expanded interleaver is also a structure where the mappings are restricted to two separate partitions of the input sequence. In the case of the odd-even partitioning in the previous section, the mappings were constrained to be from one set to the other. We will now study the benefits of instead requiring the mappings to be within each of the two sets. The advantage of this restriction is that the resulting interleaver can be viewed as a combination of two separate interleavers, each half the size of the combined interleaver. Assume that the two sets consist of the odd and even positions respectively. Then, by using every second element in the stored interleaver vector, an interleaver half as large as the original interleaver is achieved. Naturally,

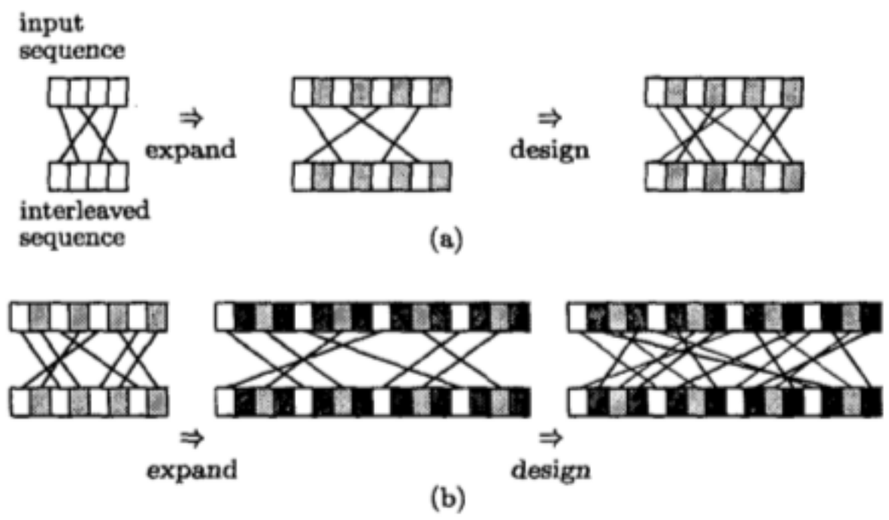


Fig. 4. (a) Expansion of a 4-bit interleaver to an 8-bit interleaver, and (b) Expansion of the 8-bit interleaver to a 16-bit interleaver. Both the original 4-bit and the intermediate 8-bit interleavers can be retrieved from the final 16-bit interleaver.

it is possible to require that also the mappings of the new, smaller interleaver are restricted to additional, similar, partitions. As long as this requirement is fulfilled, smaller and smaller interleavers can be produced, by simply using every second element in the nearest larger interleaver.

A straightforward way of constructing a series of expanded interleavers is to start with the smallest desired interleaver size, and expand this by inserting undefined elements between the already existing positions in the interleaver. The new, undefined positions are then assigned with mappings according to some interleaver design criteria, independent of the restrictions imposed by the expansion structure. After the expanded interleaver is designed, further expansions can be performed until the largest desired interleaver size is obtained. Two such expansions are shown in Fig. 4, first from a 4-bit interleaver to an 8-bit, and then from the 8-bit to a 16-bit interleaver.

The advantage of expanding interleavers as described is the reduction of storage memory required for the interleaver rules; the expanded 16-bit interleaver in Fig. 2(b) can be used to interleave 4-bit, 8-bit and 16-bit sequences. This interleaver structure is thus useful when a range of interleaver sizes are to be stored. The amount of storage memory saved approaches 50%, as the number of interleaver sizes increases.

C. Expanded odd-even symmetric interleavers

The described structures can be merged into a combined interleaver structure which is both expanded and odd-even symmetric. Due to the contradictory constraints on the odd/even subsets, the expansion need to be modified compared to the description above. One way to maintain the odd-even property in an expanded

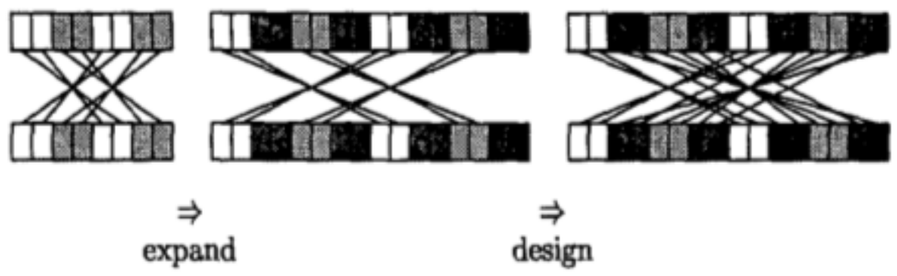


Fig. 5. An interleaver expansion that preserves the odd-even symmetric properties of an interleaver. Instead of inserting one unassigned element after every original interleaver entry, two unassigned elements are inserted after every second original entry.

interleaver is to insert *two* undefined elements after *every second* entry in the original interleaver. This modification ensures that each element on an even position in the original interleaver remains on an even position after expansion, and vice versa. Thus, if the original interleaver meets the odd-even symmetric constraints, and if the assignment of the inserted elements is performed with these restrictions, the new interleaver will be an *expanded, odd-even symmetric interleaver*. The expansion of an 8-bit odd-even symmetric interleaver to a 16-bit odd-even symmetric interleaver is illustrated in Fig. 5.

The expanded and designed 16-bit interleaver in Fig. 5 can be stored on hardware as $\tilde{\pi}_{16}$=[5 8 7 6 1 2 3 4]. For these small interleaver sizes, the restrictions imposed on the interleavers reduce the design freedom substantially. However, when designing interleavers with more reasonable sizes (at least one hundred bits), the imposed restrictions do not severely reduce the design freedom of the interleavers.

The implementation of expanded odd-even symmetric interleavers is identical to the odd-even symmetric implementation – with the addition that the same hardware can be used for a range of interleaver sizes. Interleaving of 16-, 8- and 4-bit sequences with the interleaver in Fig. 5 are illustrated in Fig. 6. The only difference between interleaving an 8-bit sequence instead of a 16-bit sequence is that the interleaver index-counter is clocked twice between each new bit to be stored/read. This holds also for the index-counter for the sequentially stored/read bits, which now use addresses $1, 3, 5, \ldots$, instead of $1, 2, 3, \ldots$.

III. Simulation results

The error correcting performances of turbo codes using interleavers with the described structures have been simulated for AWGN channels. We use rate-1/3 turbo codes with up to 15 de-

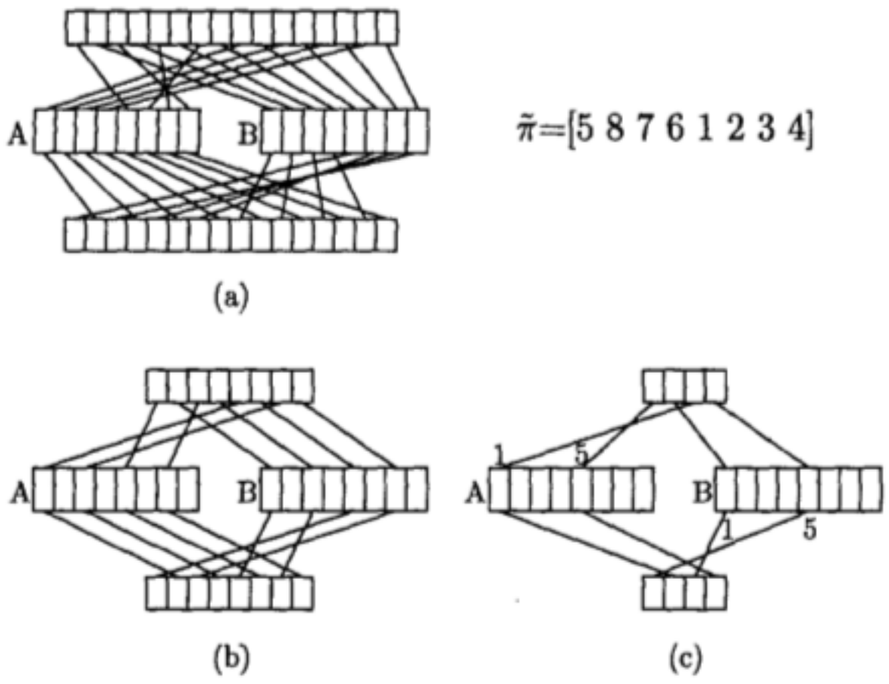


Fig. 6. Implementation of a 16-bit expanded odd-even symmetric interleaver, used to interleave (a)16-bit-, (b) 8-bit- and (c) 4-bit input sequences. For each step down in input sequence length, the clock speed of the index counter for $\tilde{\pi}$ is doubled. For example, when interleaving the 4-bit sequence, the only addresses used in $\tilde{\pi}$ is 5 and 1.

coding iterations, where each constituent decoder employ the log-MAP decoding algorithm. The generator polynomials of the constituent encoders were $(1, 17/15)_{\text{oct}}$. The first encoder was terminated with tail bits, while the second encoder was truncated in an unknown state.

Simulations were performed with interleavers of sizes 320, 640, 1280, 2560 and 5120 bits. The 320-bit interleaver is an odd-even symmetric interleaver, while all the rest are expanded, odd-even symmetric. Aside from these interleaver restrictions, the interleavers were designed to yield good correlation properties of the extrinsic information [9], as well as good distance properties of the resulting turbo codes. The bit- and frame-error rate results after 8 and 15 decoding iterations are shown in Fig. 7 and Fig. 8 respectively. In these figures, the structured interleavers are denoted EOES-CDI; expanded odd-even symmetric, correlation designed interleaver. As a performance reference, we use interleavers designed completely without structure restrictions, denoted CDI. This type of interleavers have shown very competitive performances in comparison to interleavers designed by many other design strategies, see for example [9], [10]. When only one line is visible in the performance plots, it is because the performances are right on top of each other. The results indicate that the presented interleaver structures have essentially no influence on the performance of the compared turbo codes, as long as the interleavers

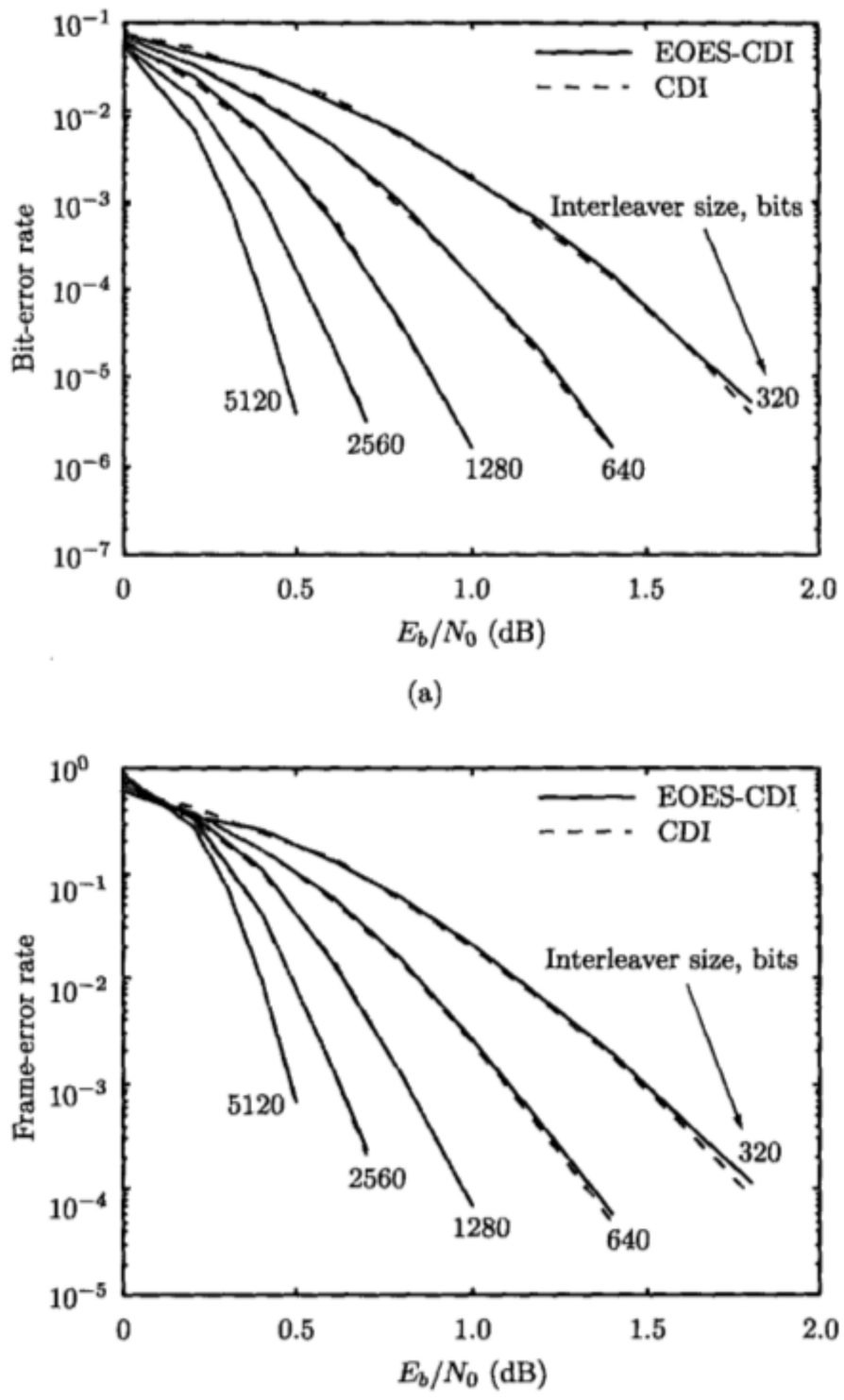


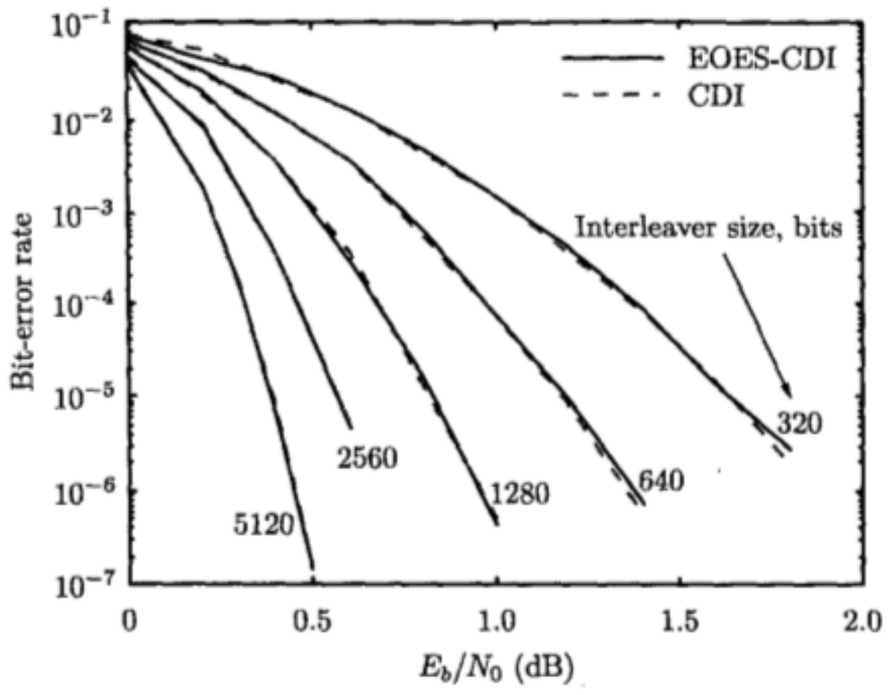
Fig. 7. Simulated bit- and frame-error rates after 8 decoding iterations of rate-1/3 turbo codes of various interleaver sizes, on an AWGN channel. The expanded odd-even symmetric (EOES-CDI) interleavers perform essentially as well as the interleavers designed entirely without the structure restrictions (CDI).

are properly designed.

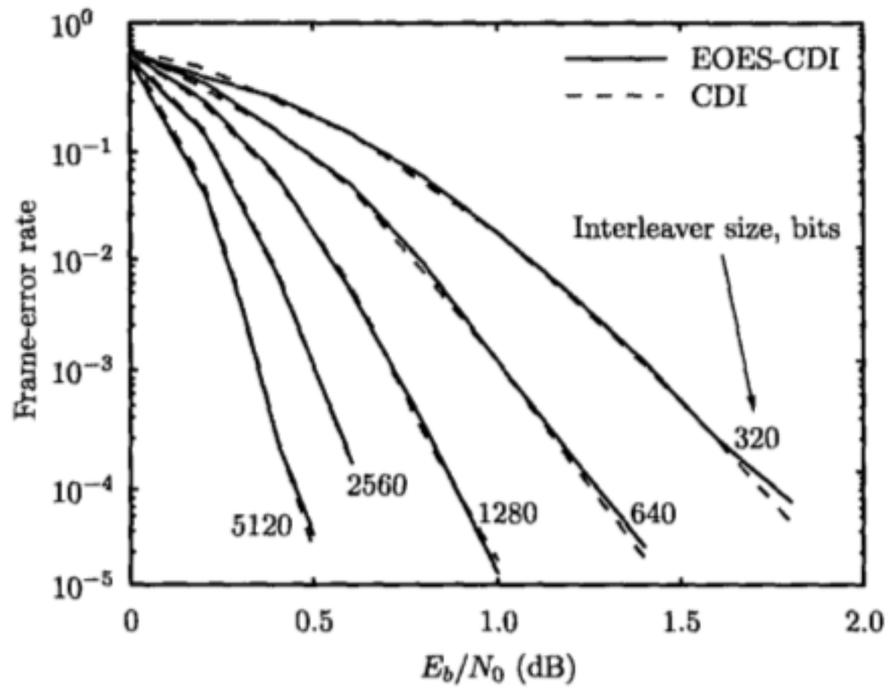
We present here only the performances of interleavers designed with both the expanded and the odd-even symmetric restrictions. However, interleavers designed with only one of the constraints, as presented in Section II-A and II-B respectively, perform the same as the performances shown in Fig. 7 and Fig. 8.

IV. Discussion and conclusions

The concept of pruning interleavers were discussed in [7]. When pruning interleavers, very high interleaver size granularity is achieved by disregarding the interleaver mappings to posi-



(a)



(b)

Fig. 8. Simulated bit- and frame-error rates after 15 decoding iterations, of rate-1/3 turbo codes of various interleaver sizes, on an AWGN channel. The expanded odd-even symmetric (EOES-CDI) interleavers perform essentially as well as the interleavers designed entirely without the structure restrictions (CDI).

tions above the size of the desired interleaver. For example, a 570-bit interleaver is obtained by disregarding all the elements that are larger than 570 in the nearest larger interleaver, i.e. the 640-bit interleaver in our case. However, for the interleavers evaluated in this paper we found that the correlation properties of the extrinsic information are better preserved, if the pruning is performed at both ends of an interleaver. This means that elements both below and above a certain number, depending on the desired interleaver size, are disregarded.

The storage of an expanded odd-even symmetric interleaver of size 5120 bits requires 2560 addresses to be stored, each address represented by 12 bits. This amounts to approximately $3.1 \cdot 10^4$ memory cells. As described, this interleaver can be used to interleave all sequences with lengths on the form $5120/2^k$, $k = 0, 1, 2, \ldots$, down to the size of the original interleaver used for the first expansion. Assuming instead that interleavers with sizes ranging from 320 to 5120 bits are to be stored using unstructured interleavers, these interleavers require a total of $\sum_{k=0}^{4} 320 \cdot 2^k \left\lceil \log_2 \left(320 \cdot 2^k \right) \right\rceil \approx 1.2 \cdot 10^5$ memory cells for storage. The reduction using the expanded odd-even symmetric interleaver is thus 74% in terms of storage area. Furthermore, switching between interleaver sizes is very easily implemented by shifting the bits in the interleaver index counter to the left, one position for each step down in interleaver size. Simulations show that the presented interleaver constraints have essentially no influence on the error correcting performances of the codes.

References

[1] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-Codes," *IEEE Transactions on Communications*, vol. 44, pp. 1261–1271, October 1996.

[2] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Transactions on Communications*, vol. 44, pp. 591–600, May 1996.

[3] S. Benedetto, R. Garello, and G. Montorsi, "A search for good convolutional codes to be used in the construction of turbo codes," *IEEE Transactions on Communications*, vol. 46, Sep. 1998.

[4] D. Divsalar and R. J. McEliece, "Effective free distance of turbo codes," *Electronic Letters*, vol. 32, Feb. 1996.

[5] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations." TDA progress report 42-122, Jet propulsion Lab., Pasadena, CA, August 1995.

[6] S. Crozier, J. Lodge, P. Guinand, and A. Hunt, "Performance of turbo codes with relative prime and golden interleaving strategies," in *Sixth International Mobile Satellite Conference*, Ottawa, Canada, June 1999.

[7] M. Eroz and A. R. Hammons, "On the design of prunable interleavers for turbo codes," in *Vehicular Technology Conference*, Houston, USA, May 1999.

[8] A. S. Barbulescu and S. S. Pietrobon, "Terminating the trellis of turbo-codes in the same state," *Electronics Letters*, vol. 31, pp. 22–23, January 1995.

[9] J. Hokfelt, O. Edfors, and T. Maseng, "Interleaver design for turbo codes based on the performance of iterative decoding," in *IEEE International Conference on Communications*, Vancouver, BC, Canada, June 1999.

[10] A. Henriksson, J. Hokfelt, and O. Edfors, "Evaluation of an interleaver design algorithm for turbo codes in UMTS." Tdoc 419/98, ETSI SMG2 UMTS L1 Expert Group, Meeting no 7, Sweden, Oct. 1998.