Lossless Turbo Source Coding with Decremental Redundancy

Joachim Hagenauer, João Barros, Andrew Schaefer
Lehrstuhl für Nachrichtentechnik (LNT)
Technische Universität München (TUM)
D-80290 München, Deutschland
Email: {hag,joao,andrew}@lnt.ei.tum.de

Abstract

Recent results indicate that the same turbo principle which delivers near to optimal strategies for channel coding, can be used to obtain very efficient source coding schemes. We investigate this issue applying ten Brink's EXIT chart analysis and show how this technique can be used to select the most efficient match of component codes and puncturing matrices to compress discrete memoryless sources. Aiming at perfect reconstruction at the decoder, i.e. lossless source coding, we present an encoding algorithm, which gradually removes the redundancy while checking the decodability of the compressed bit stream. This concept of *decremental* redundancy is dual to the principle of *incremental* redundancy that characterizes hybrid ARQ (Type II) communication protocols. Both principles can be combined when the channel is noisy.

1 Introduction

It is well known that data compression and data transmission are essentially dual problems. In compression we remove all the redundancy in the data to form the most compressed version possible – the focus of source coding – whereas in transmission over noisy channels we add redundancy in a controlled fashion to combat errors in the channel – the purpose of channel coding.

In the past decade a series of research breakthroughs in the field of channel coding (most notably [1]), have rendered a large number of coding schemes which can achieve near-to-optimal performance (i.e. very close to channel capacity) with reasonable complexity. The secret of their success lies in the use of a "turbo" feedback in an iterative decoding scheme, an ingenious approach we call the "turbo principle" [2].

More recently, the turbo principle has been applied successfully to data compression of binary memoryless sources [3]. While in [4] the objective is to improve the error correction after transmission over a channel and therefore all encoded bits are useful, in the dual source coding problem addressed in [3] there is no channel involved and so the encoded bits can be punctured heavily depending on the desired compression rate. An alternative scheme based on LDPC codes and belief propagation decoding was presented in [5].

1.1 Main Contributions and Organisation of the Paper

Our main contributions are as follows. We begin by questioning the concept of *near-lossless* source coding, that tolerates an error probability of about 10^{-5} and characterizes most of the previous contributions to this problem, [3], [5]. Instead, we propose an encoding scheme which guarantees perfect reconstruction at the decoder, i.e. lossless source coding. This encoding algorithm exploits the duality between this problem and ARQ Type II protocols. While in the latter channel coding problem we gradually transmit additional parity bits until the decoder can correct all errors induced by the channel (*incremental redundancy*, IR), in the source coding version we remove bits as long as the encoder can guarantee perfect reconstruction (*decremental redundancy*, DR).

To enable the analysis and design of lossless turbo source codes, we modify ten Brink's extrinsic information transfer ("EXIT") charts [6] to account for non-uniform binary sources. With this tool we are able to match different component codes and puncturing rates yielding source coding schemes that perform at compression rates close to the entropy.

In Section 2 we give a formal statement of the problem. The turbo source encoder and decoder are explained in Section 3, followed by a detailed discussion of the proposed decremental redundancy algorithm in Section 4. Section 5 then introduces the modified EXIT chart for turbo source coding. The paper concludes with some examples and final remarks in Sections 6 and 7, respectively.

2 Problem Statement

Let U be a discrete, memoryless source drawn i.i.d. $\sim p(u)$ from the alphabet $\mathcal{U} = \{1, 2, \ldots, L\}$. The optimal compression rate for the source U is given by the entropy

$$H(U) = -\sum_{i=1}^{L} p(i) \log p(i) \qquad (1)$$

A source encoder for source U takes a block of N symbols $U^N = U_1, U_2, \ldots, U_N$ and generates a binary codeword $X^K = X_1, X_2, \ldots, X_K$ with $X_i \in \{0, 1\}$, respectively $\{+1, -1\}$. The rate of the source code is defined as $R = K/N$. We design our source encoders based on the principle shown in Fig. 1. Each symbol in the block U^N is first mapped to a binary codeword of length $l = \lceil \log_2 L \rceil$. A turbo encoder, for which more details are provided in the next section, then takes a block of binary encoded symbols and delivers a block of codebits, which are punctured to achieve the desired compression rate.

We ask two questions: (1) how do we guarantee *lossless* source coding, i.e. perfect reconstruction of the source at the decoder and (2) how should we design the turbo code and the puncturing matrices to achieve maximal compression, i.e. to approach the entropy of the source?

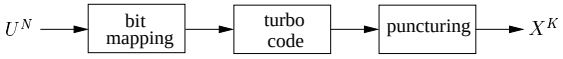

Fig. 1. An encoder for turbo source coding.

3 Turbo Source Coding (TSC) of Binary Sources

3.1 Source Encoding

Consider a block of source symbols U^N. Since the entropy of the source is $H(U)$, the source block can be perfectly reconstructed from a binary sequence X^K of length $K \simeq NH(U)$ for sufficiently large N (source coding theorem). Our task is to design a mapping from the source to the compressed sequence. We assume a binary source ($L = 2$).

The motivation for using "turbo codes" for source compression lies in the duality of source and channel coding. The latter is a sphere packing problem, whereas the former is a sphere covering problem ([8], pp. 357). It follows that a good channel code is likely to be a good source code. Since "turbo codes" are well known to be good channel codes (i.e. they provide a good solution to the sphere packing problem), it is just a natural step to apply these to source coding problems. We now proceed to discuss the question of "how".

As stated above, we wish to generate a binary sequence X^K from the source U^N. It is possible to do this as shown in Fig. 2. We pass each of the sequences U^N and $\Pi(U^N)$ (Π defines the interleaver function), through rate 1 convolutional codes (or scramblers) with feedback to generate the parity sequences P_1^N and P_2^N respectively. As opposed to parallel concatenation in channel coding, we discard the source bits since the source decoder knows of the source statistics apriori (this topic will be addressed below).

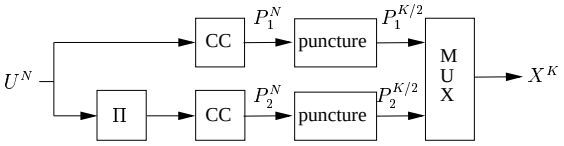

Fig. 2. A parallel concatenation of rate 1 convolutional codes (CC) with an interleaver between them followed by heavy puncturing can be used to perform data compression of binary sources.

We now have the sequences P_1^N and P_2^N, which when put together are of length $2N$ meaning that we do not yet have any compression. We achieve compression by puncturing. If we opt for a fixed but randomly chosen puncturing scheme then the puncturing operation is analogous to transmitting the parity sequences over a binary erasure channel (BEC), whose "erasure probability" can be adjusted to achieve the required compression rate. We then rely on the turbo decoder to reconstruct the original message after transmission over the dual noisy channel (in this case our BEC). If ϵ represents the proportion of erased bits, then we have

$$\epsilon \approx 1 - \frac{H(U)}{2}. \tag{2}$$

Thus the sequence X will be of length approximately $K = NH(U)$ and we have compressed our sequence close to its entropy. We now address the problem of reconstructing U^N from X^K.

3.2 Source Decoding

At first glance, random puncturing of bits does not seem to be a very sophisticated way of performing source compression. However, just as in channel coding with turbo codes, the sophisticated part lies in the use of a turbo decoder. The turbo decoder for our system is depicted in Fig. 3.

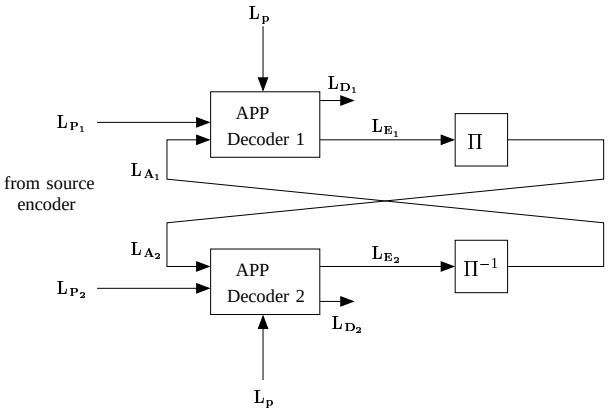

Fig. 3. Parallel turbo source decoder.

We will use L-values to describe the values input and passed around by the turbo decoder. Given a binary random variable U, where for convenience $u \in \{+1, -1\}$, then the L-value is defined as,

$$L(U) = \log_e \frac{P(u=+1)}{P(u=-1)}. \quad (3)$$

More details on L-values can be found in [9]. In Fig. 3, $\mathbf{L_{P_1}}$, $\mathbf{L_{P_2}}$, $\mathbf{L_{D_1}}$, $\mathbf{L_{D_2}}$, $\mathbf{L_{A_1}}$, $\mathbf{L_{A_2}}$, $\mathbf{L_{E_1}}$ and $\mathbf{L_{E_2}}$ represent sequences of L-values. The letters A and E stand for "apriori" and "extrinsic" respectively. Since the parity bit sequences $P_1^{K/2}$ and $P_2^{K/2}$ can be thought of as being "transmitted" through a binary erasure channel (i.e. punctured), the corresponding input sequences $\mathbf{L_{P_1}}$ and $\mathbf{L_{P_2}}$ take on the L-values $\pm\infty$ (perfectly known bits) or 0 (erased).

In the case of a non-uniform binary source with $P(U=+1) = p$ and $P(U=-1) = 1-p$, for which $H(U) = H_b(p)$ (i.e. the binary entropy function), each decoder has an additional input vector $\mathbf{L_p}$, where each element of the vector is equal to $L_p = \log_e\left(\frac{p}{1-p}\right)$. We call this 'source' apriori knowledge and distinguish between apriori knowledge which is learnt during the iterations of the decoder which we will call 'learnt' apriori knowledge. In the case of more sophisticated (Markov) source models our source apriori vector would contain different sets of L_p values. The decoding algorithm is carried out according to Fig. 3. APP decoder 1 uses $\mathbf{L_{P_1}}$ and $\mathbf{L_{A_1}}$ (initialised to zero in the first iteration) to calculate the extrinsic L-values $\mathbf{L_{E_1}} = \mathbf{L_{D_1}} - \mathbf{L_{A_1}} - \mathbf{L_p}$. The sequence $\mathbf{L_{E_1}}$ is interleaved and passed onto APP decoder 2 as "learnt" apriori knowledge $\mathbf{L_{A_2}}$. Decoder 2 performs analogous operations and the process is then "iterated" until covergence is achieved. Note that convergence is in general not guaranteed and this is discussed in the next section.

4 Decremental Redundancy for Lossless Turbo Source Coding

The iterative decoding algorithm will not converge if we puncture more bits than can be corrected by the chosen turbo code. Clearly, without convergence of the decoding result the source information cannot be perfectly recovered. This form of operation is sometimes called "near-lossless" source coding, indicating that some residual errors remain after decoding. Although this approach was the one adopted in the original contribution of [3], it renders turbo source coding unsuitable for the large number of applications in which the information must be recovered without flaws.

To guarantee that the decoder is able to decode the input sequence without errors (lossless source coding), we can enable the encoder to test the decodability of its output. Since we want to achieve as much compression as possible, we puncture the parity bits on a step by step basis (*decremental redundancy*) as long as the compressed block can still be decoded error free by the test decoder at the source compression side.

TABLE I
DUALITY BETWEEN INCREMENTAL REDUNDANCY (IR) CHANNEL CODING (ARQ TYPE II) AND DECREMENTAL REDUNDANCY (DR) SOURCE CODING

	IR-Channel Coding	DR-Source Coding
Redundancy Handling	Increment redundancy at each transmission, decoding trials at the receiver	Decrement redundancy at each compression, decoding trials at the transmitter
A priori information at both sides	Sequence of Puncturing Tables	Sequence of Puncturing Tables
Required State Information	Channel state information L_c for the channel decoder	Source state information L_p for the source decoder
Side Information	Successful decoding flag (ARQ) transmitted to sender	Successful encoding rate (parity index i) transmitted to decoder
Error Detection	CRC	Comparison to data sequence
Figure of Merit	Average throughput	Average compression
System Failure	if CRC fails	if decodability test at compressor side fails
Main Complexity	at the decoder	at the encoder
Codes	coarsely punctured RCPC codes or RCP Turbo Codes	finely punctured parallel concatenated systematic rate 1 convolutional codes
Decoder	A-priori Viterbi Algorithm or APP decoder	APP decoder
Extensions	less redundancy if source statistics are known	more redundancy if channel errors are expected

The approach we propose is dual to a hybrid automatic repeat request (ARQ Type II) scheme with forward error correction (FEC) [10], in which parity bits are transmitted successively (*incremental redundancy*) to build up a code which is powerful enough to correct all errors introduced by the communications channel. The duality between incremental redundancy channel coding and our proposed decremental redundancy source coding scheme is illustrated in Table I. In [10] the problem is solved using rate-compatible punctured convolutional (RCPC) codes, which result from periodic puncturing of low-rate convolutional codes. The encoder typically processes blocks of 100 to 1000 information bits and uses a puncturing pattern with period 8 and rates with incremental redundancy, e.g. $8/9, 8/10, \ldots, 8/24$.

For turbo source coding we require block lengths of about $N = 10^4 \ldots 10^6$ and finer puncturing steps are needed than those achieved with periodic puncturing with period 8. For example, if we puncture every 4th and then every 8th bit of a block, we get a compression rate of first $R = 0.875$ and then $R = 0.750$. This corresponds to a puncturing step of 0.125, which is too coarse for fine-tuning our turbo source encoder at compression rates close to entropy.

Therefore we must find another type of puncturing

scheme that works for large blocks. In summary, a puncturing scheme that is suitable for the turbo source coding problem must fulfill the following requirements:

1) the puncturing steps must be small enough to allow the compression rates to come close to the entropy,
2) the punctured parity bits should be spread out in the block to guarantee successful decoding (avoid long erasure bursts),
3) the side information, i.e. the extra bits needed for the decoder to identify the positions of the punctured bits, should be negligible in comparison to block length.

4.1 An Algorithm for Decremental Redundancy

We propose the decremental redundancy scheme illustrated in Fig. 4. In a first step the parity bits in each block P_i^N with $i = 1, 2$ are written line by line in a matrix $N_c \times N_c$, such that $N = N_c^2$. We call each column of the matrix a parity segment and index it by $i \in \{1, \ldots, N_c\}$. To gradually remove the redundancy

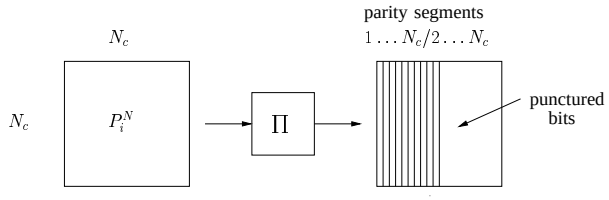

Fig. 4. Puncturing scheme for decremental redundancy.

in the encoded stream we can eliminate one such parity segment at a time (in parallel for P_1^N and P_2^N), yielding a puncturing step of $2N_c/N$ that allows us to fine tune the compression rate. To make sure that the erased bits are spread out in the block we interleave the parity bits before puncturing them. Notice that we can use the same interleaver as for the turbo code, i.e. Π and Π^{-1} for P_1^N and P_2^N, respectively, thus avoiding the storage of extra interleavers for the parity bits.

Since the decoder on the receive side knows the block length, the random puncturing matrix, and the interleaver, all it requires is the index of the last parity segment that has been erased. Since at least half the parity bits are punctured (otherwise we would have no compression at all), it suffices to use $\lceil \log_2(N_c/2) \rceil$ bits as side information (e.g. if $N = 10^4$ bits, then we get a puncturing rate step of 0.02 and require only 7 bits of side information per encoded block).

We now proceed to describe our decremental redundancy algorithm for turbo source encoding, which has the following steps:

1) Let $i := N_c/2$.
2) Encode the source block with a turbo encoder and store the output block.
3) Puncture the encoded block using i parity segments.
4) Decode the compressed block.
5) Check for errors. If the decoded block is error free, let $i := i - 1$ and go back to 3.
6) Let $i := i + 1$. Repeat 3, include a binary codeword corresponding to the index i, and stop.

Notice that the number of iterations required to find the best puncturing rate can be considerably reduced, if we store a table with different starting points for the optimization algorithm, depending on the probability distribution of the source. Since the sequences produced by the source are likely to be typical [8] the number of iterations required to fine tune the compression rate will in general be small. We can also turn to a more sophisticated segmentation of the parity matrix, depending on the desired puncturing step.

4.2 Combining Decremental and Incremental Redundancy for Joint Source and Channel Coding

It is worth noting, that the previous algorithm can be used both for decremental *and* incremental redundancy depending on whether the application requires pure data compression or robust transmission over a noisy channel. With very mild modifications, the algorithm allows us to vary the redundancy in the output bit stream almost continuously to match the entropy of the source or the capacity of the channel.

For example, if channel state information is available at the transmitter, we can add a test channel to the encoding loop that t ests the decodability of the encoded block. Then, if the decodability check fails, the encoder simply adds more parity segments and proceeds with the next iteration. Similarly, if a feedback channel is available, the receiver can send an automatic repeat request every time it fails to decode a block and the encoder can send extra parity segments in the spirit of ARQ Type II protocols where an extra CRC checks the integrity of the source data.

The described coding strategy is particularly elegant, because independently of the source coding or channel coding application, the core turbo code remains exactly the same and all we vary is the number of punctured bits, thus fixing the encoding rate in a very flexible way.

5 EXIT Charts

To analyse the convergence and thus obtain the achievable compression rates (or the starting points) for our turbo source coding algorithm, we can employ slightly modified versions of the standard tools used for optimizing turbo codes.

A useful tool for analysing iterative decoding algorithms is the EXIT chart (e.g. [11],[12]). The principle behind the EXIT chart is that much can be learned about the overall performance of a turbo decoder by

analysing each of its parts. For this purpose, EXIT charts use mutual information to parameterise the sequences of L-values being exchanged between decoders and thus characterise the APP decoders.

In this section, we show how to modify the EXIT chart for use in our turbo source coding problem. We start by looking at the mutual information between a continuous random variable and a non-uniform binary source.

5.1 Mutual Information for Non-Uniform Binary Sources

Assume again a binary source U drawn i.i.d. from the alphabet $\mathcal{U} = \{+1, -1\}$, such that $P(U = +1) = p$.

According to the channel coding formulation of the EXIT chart, the encoded block is transmitted over a channel with probability density function $f(y|U = u)$, where y denotes the channel output. Because the source is assumed to be uniform, $f(y|U = u)$ is usually symmetric. We can write the mutual information between the random variables U and Y as,

$$I(U;Y) = \sum_{u=\pm 1} p(U=u) \int_{-\infty}^{+\infty} f(y|U=u) \cdot \log_2 \frac{f(y|U=u)}{p \cdot f(y|U=+1) + (1-p) \cdot f(y|U=-1)} dw, \quad (4)$$

Following [13], which exploits the ergodicity of the source this expression can be simplified to

$$\begin{aligned} I(U;Y) &= H_b(p) - E\{\log_2(1 + e^{-u \cdot L(U|Y)})\} \\ &\simeq H_b(p) - \frac{1}{N} \sum_{n=1}^{N} \left(1 + e^{-u_n \cdot L(u_n|y_n)}\right) \end{aligned} \quad (5)$$

where the expectation is taken over all possible observations. Note that when no information is transmitted across the channel we have $L(U|Y) = \log_e \frac{p}{1-p}$ which results in $I(U;Y) = 0$ as expected. Thus, we are using our knowledge of the source statistics in the equation for the calculation of the mutual information.

This mutual information measure is now used to analyse the turbo decoder. Particularly useful will be $I(U;\mathbf{L_{E_1}})$, $I(U;\mathbf{L_{A_2}})$, $I(U;\mathbf{L_{E_2}})$ and $I(U;\mathbf{L_{A_1}})$ which we will from now on denote as I_{E1}, I_{A2}, I_{E2} and I_{A1} respectively.

5.2 Characteristic Curves

We now characterise the component decoders of the turbo decoder by determining their extrinsic information transfer characteristics (EXIT charts, also called characteristic curves). The aim is to determine $I_{E1} = f_1(I_{A1})$ for decoder 1 and $I_{E2} = f_1(I_{A2})$ for decoder 2.

The procedure we use to obtain the characteristic curves is illustrated in Fig. 5. First, we encode a binary

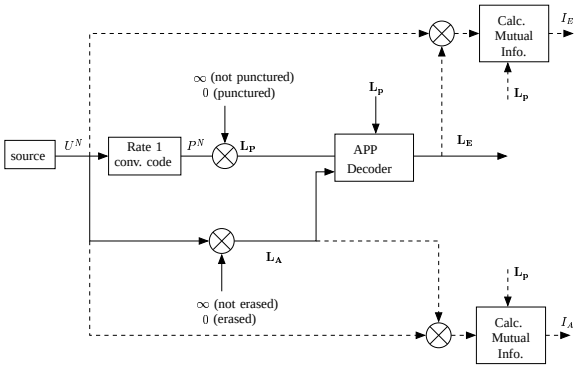

Fig. 5. Determining characteristic curves of component decoders. The encoder is a rate 1 feedback convolutional encoder, also called *scrambler*. Note the dashed arrows indicating where mutual information should be measured according to equation (5).

sequence U^N with $P(U = +1) = p$ using a rate 1 convolutional encoder (*scrambler*) and puncture the resulting parity bits P^N with a puncturing rate ϵ (this may be thought of as a BEC with erasure probability ϵ). We then run the APP decoding algoritm with various amounts of apriori information for the source bits by generating appropriate sequences of L-values $\mathbf{L_A}$.

Referring to figure 5, we see that $\mathbf{L_A}$ is generated by passing the sequence U through an "apriori channel" which we model as a BEC with erasure probability δ. To calculate I_A we apply equation 5. Note that we must now also use our knowledge of the source statistics to calculate the mutual information.

Based on the L-values $\mathbf{L_P}$ of the punctured parity bits and the a priori sequence $\mathbf{L_A}$, the APP decoder generates an output sequence $\mathbf{D}$, from which we obtain the extrinsic L-value sequence $\mathbf{L_E} = \mathbf{L_D} - \mathbf{L_A} - L_p$. We again use equation (5) together with the source statistics to generate I_E. Note that $\mathbf{L_E}$ and $\mathbf{L_A}$ correspond to the sequences of L-values which are passed between the component decoders of Fig. 3.

5.3 Decoding Trajectory and Construction of the EXIT Chart

Given that in the generation of the characteristic curves, the apriori sequences are modelled accurately (in our case we used a BEC), it is possible to predict the convergence of an iterative decoder without having to simulate the iterative decoder itself. This is one of the main advantages of the EXIT chart. Exactly how this is done is discussed in the next section. In our case, convergence can be interpreted as the ability of the turbo decoder to rebuild the source sequence from the compressed sequence.

To construct the EXIT chart, we start by noting that $I_{E1} = I_{A2}$ and $I_{E2} = I_{A1}$. Thus we can plot $I_{E1} = f_1(I_{A1})$ against a mirrored version of $I_{E2} = f_2(I_{A2})$ (we mirror on the 45^o axis). If a tunnel exists between the two curves, then the sequence can be successfully decompressed. The width of the tunnel

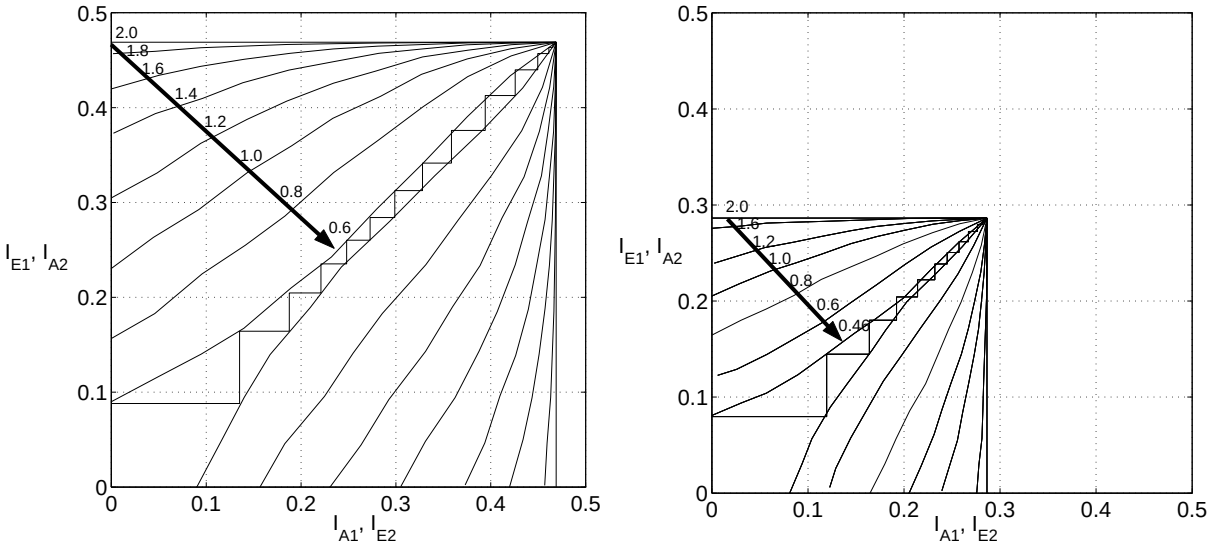

Fig. 6. EXIT charts for two binary sources with $H(U) = 0.469$ (left) and $H(U) = 0.286$ (right), $N = 9 \cdot 10^4$, parallel concatenation of two convolutional codes with polynomials [7, 5]. Each EXIT chart corresponds to a different total encoding rate, achieved by gradually puncturing more bits and thus removing the redundancy. As long as there is an open tunnel for the decoding trajectory, the convergence of the turbo source decoding algorithm is guaranteed.

can be changed by changing the compression rate (or equivalently puncturing rate) of the source encoder.

In section 6, we verify that the EXIT chart predicts indeed the behaviour of the source decoder by comparing it against an actual decoding trajectory. The result supports our claim that our procedure does measure the relevant mutual informations in the decoder during a decoding run and therefore the modified EXIT charts we present, can be used to find the best match of component codes and puncturing rates for the turbo source coding problem.

6 Examples

To underline the effectiveness of the proposed source coding tools, we present a set of numerical results, which further illustrate the concept of decremental redundancy and gives an idea of the performance one can expect from parallel concatenated turbo source codes. Fig. 6 shows EXIT charts and decoding trajectories for two binary sources (i.i.d.) with entropies 0.469 and 0.286 (on the left and right plots, respectively). To obtain the trajectory, we used a block of $N = 9 \cdot 10^4$ bits. This is equivalent to averaging the trajectories of 9 of our test blocks of size $N = 10^4$ (see below). Note that the decoding trajectory of each individual, relatively small block of 10^4 bits does not necessarily match the characteristic curves but the average of many blocks does.

As outlined in the decremental redundancy algorithm we successively remove parity segments, allowing the compression rate to gradually approach the entropy. Notice that the width of the square containing the EXIT chart is equal to $H_b(p)$, as opposed to the unitary square typical of channel coding EXIT charts. The trajectory shows the actual exchange of information between the two decoders, which remains possible only if the tunnel between the two corresponding curves is open. These modified EXIT charts can also be used to choose the component codes for the turbo source code, as shown in Fig.7. We can see that in the source coding case increasing the memory of the component convolutional codes does not improve the performance significantly.

To illustrate the performance of the proposed turbo source coding scheme, Fig. 8 shows a normalized histogram of the achieved compression rates for 1000

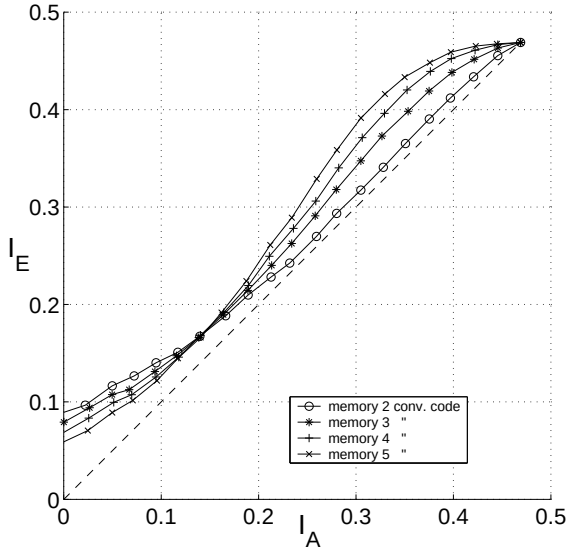

Fig. 7. EXIT charts for component codes with different memories and a source with entropy $H(U) = 0.469$ and total encoding rate of $R = 0.6$.

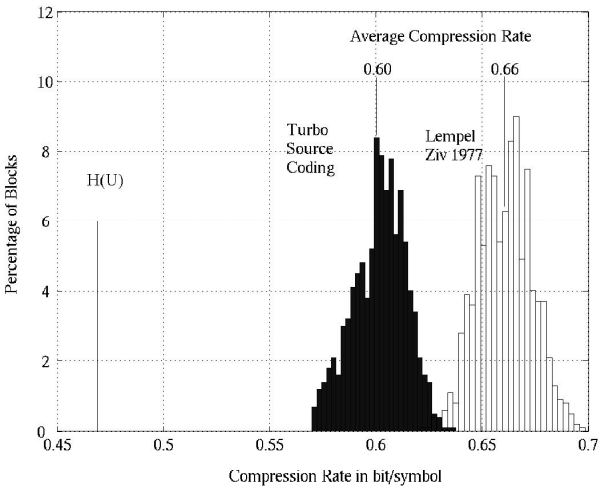

Fig. 8. Normalized histograms to illustrate the performance of turbo source coding versus lempel-ziv compression. The values were obtained by compressing 1000 blocks of $N = 10^4$ symbols from a binary source with $H(U) = 0.469$ using less than 10 iterations.

TABLE II

PREDICTED AVERAGE COMPRESSION RATES R FOR TURBO SOURCE CODING OF BINARY SOURCES($N = 9 \cdot 10^4$) FROM EXIT CHARTS.

p	0.01	0.05	0.1	0.15	0.2	0.3
$H_b(p)$	0.081	0.29	0.47	0.61	0.72	0.88
R	0.25	0.44	0.59	0.71	0.8	0.98

blocks of $N = 10^4$ i.i.d. samples from a binary source of entropy $H(U) = 0.469$. A comparison with the wide spread Lempel-Ziv 1977 algorithm (universal source coding) shows that for the same block length turbo source coding yields lower compression rates, i.e. closer to the entropy of the source.

Finally, Table II summarizes the results obtained for binary sources with different entropies. The shown average compression rates were obtained for individual source blocks by plotting the EXIT charts with decremental redundancy as long as the tunnel was open. Notice that the average compression rate predicted by the EXIT chart is very close to the experimental results illustrated in Fig. 8.

7 Conclusions

We have presented a turbo source coding scheme, which guarantees lossless recovery of the source information through a decremental redundancy loop in the encoder. The convergence of the turbo source decoder can be analyzed using a modified EXIT chart, which becomes a powerful tool to optimize the component codes and the puncturing rates near the entropy of the source. Possible directions for future work include the design of irregular turbo source codes with EXIT charts [13], application to more elaborate discrete sources and entropy-constrained quantization of continuous-valued sources using turbo source codes.

Acknowledgements

The authors gratefully acknowledge useful discussions with Michael Tüchler. Some of the computer programs were written by Jian Shen as part of his master's thesis at TU München.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error–correcting coding and decoding: Turbo-codes," in *Proc. IEEE International Conference on Communications (ICC)*, (Geneva, Switzerland), pp. 1064–1070, May 1993.

[2] J. Hagenauer, "The turbo principle - tutorial introduction and state of the art," in *Proc. Int. Symposium on Turbo Codes & Related Topics*, (Brest, France), pp. 1–11, 1997.

[3] J. Garcia-Frias and Y. Zhao, "Compression of Binary Memoryles Sources Using Punctured Turbo Codes," *IEEE Communications Letters*, pp. 394–396, 2002.

[4] J. Hagenauer, "Source–controlled channel decoding," *IEEE Transactions on Communications*, vol. 43, pp. 2449–2457, September 1995.

[5] G. Caire, S. Shamai, and S. Verdu, "A new data compression algorithm for sources with memory based on error correcting codes," in *Proc. IEEE Information Theory Workshop*, (La Sorbonne, Paris, France), pp. 291–295, March 31 - April 4 2003.

[6] S. Brink, "Designing iterative decoding schemes with the extrinsic information transfer chart," *AEÜ — International Journal of Electronics and Communications*, vol. 54, no. 6, pp. 389–398, 2000.

[7] J. Garcia-Frias and Y. Zhao, "Compression of Correlated Binary Sources Using Turbo Codes," *IEEE Communications Letters*, pp. 417–419, 2001.

[8] T. M. Cover and J. Thomas, *Elements of Information Theory*. John Wiley and Sons, Inc., 1991.

[9] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, pp. 429–445, March 1996.

[10] J. Hagenauer, "Rate–compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Transactions on Communications*, vol. 36, pp. 389–400, April 1988.

[11] S. Brink, "Iterative decoding trajectories of parallel concatenated codes," in *Proc. 3rd ITG Conference Source and Channel Coding*, (Munich, Germany), pp. 75–80, January 2000.

[12] S. Brink, "Design of serially concatenated codes based on iterative decoding convergence," in *Proc. 2nd Int'l Symp. on Turbo codes & Related topics*, (Brest, France), pp. 319–322, September 2000.

[13] M. Tüchler and J. Hagenauer, "Exit charts for irregular codes," in *Proc. 36th Annual Conf. Inform. Sciences Syst. (CISS)*, (Baltimore, MD), 2002.