Proceedings of the 6th Nordic Signal Processing Symposium - NORSIG 2004
June 9 - 11, 2004
Espoo, Finland

Decoding of punctured turbo codes using dual codes

Kalle Ruttik
Helsinki University of Technology, Communications Laboratory,
P.O. Box 2300, FIN-02015 HUT, Finland
Kalle.Ruttik@hut.fi

***Abstract*—In this paper we show how to construct trellis of the dual code for punctured systematic recursive convolutional code. Such codes are often applied as constituent codes of turbo codes. Since they are used in wireless communication standards their efficient decoding has practical applications.**
It has been long known that decoding of high rate codes is more efficient in dual space than in normal space. For decoding in dual space one needs to generate the trellis of the dual code. In this paper we show how to generate it by finding dual code for the sections of the initial code.
The generated dual decoder performance is tested by simulations against logMAP and maxlogMAP algorithms in normal code space.

I. Introduction

Decoding of high rate turbo codes is hampered by the decoder complexity. High rate turbo code can be generated by puncturing a low rate code. In this paper we show how one can use a simple decoding algorithm in dual space for decoding the punctured code.
It has long been known that if the dual code has less codewords than the original code then decdoing is simpler if to use codewords of the dual code [1] [2].
Since the soft output MAP algorithm exists also for the dual space decoder one can implement the turbo decoder in the dual space [3].
In its initial version the dual code based decoding required identification of the dual code for the whole codeword. In [4] was shown that the decoding can significantly be simplified by dualisizing the sections of the code. In the trellis code the section is identified by the input state, transition in the trellis, and the output state. These are also sufficient to calculate the dual code for the trellis section.
In this paper we show a simple method for generating dual code trellis for the punctured recursive convolutional code. Such codes are commonly used as constituent codes for generating high rate turbo codes. By applying the dual code based decoder we are able to use simple algorithm for decoding such turbo codes.
We compare the performance of the dual decoder with performance of maxlogMAP and logMAP algorithm in normal code.
Through the paper we use as an example a $1/3$ rate PCCC systematic turbo code generated by $[\ 7 \quad 5\ ]$ generator polynomials. (figure 1) The two component encoder outputs are punctured by the same pattern $[\ 0 \quad 0 \quad 0 \quad 1\ ]$.

The paper is organized as following: in section 2 we first describe how to construct the dual code generator matrix for the particular code. The decoding in dual space is explained in section 3. The performance of the algorithm is illustrated by the simulations in section 4.

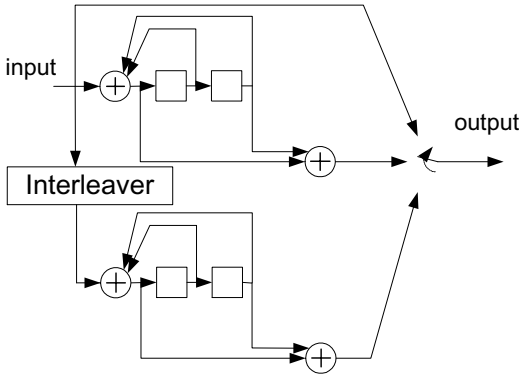

Fig. 1. Block diagram of the encoder

II. Description of the punctured convolutuonal code trough sectionwise generator matrix

We assume low rate binary code generated by two encoders with feedback polynomial. The encoders generate 1 output bit for each input bit and the inputs to different encoders are interleaved (figure 1).
From the encoder output we generate a high rate code by puncturing some of the encoded bits.
A common approach to decode a punctured code is to use the initial trellis of the nonpunctured code. On this trellis the MAP algorithm is evaluated by using loglikelihood 0 for the absent encoded bits. For the $1/2$ rate code that means in some stages in the decoder trellis the transitions have only information about the systematic bit. If to merge such subsequent trellis stages, where the encoded bit is punctured, we will have transitions that contain sequence of systematic bits ending with one encoded bit .
The trellis section (branch group $(s,(w),s')$) is completely described by the initial state s, transitions in the trellis w, and final state s'. The dual code for the branch group is orthogonal to all possible values in the branch group.

The dual code for the initial code can be calculated by finding the dual code for each of its branch groups and merging the subsequent branch groups by corresponding border conditions [4].

For calculating the dual code we need the code generator matrix for the trellis section: from one stage to the next stage. One can build such matrix by describing whether each input bit and initial state contribute to the encoded bits and output state in this trellis section.

The output encoded bits are based on the initial memory content of the encoder and on the values of the input bits. Those variables are combined to the output value accordingly to the generator polynomial. If one would like to calculate the output not for the current moment but for the next, one just has to shift the state of the encoder and compute the output. For the linear code one can calculate the output for each input bit separately and sum them together later.

For example if the encoder has two input bits with values x_1 x_2 and the generator polynomial is $\frac{G_1(D)}{G_0(D)}$. The output at time moment t is multiplication of the polynomial describing the input $x_1D^{-1}+x_2D^{-2}$ with the generator polynomial t times and reading the output at time moment t.

$$\left[x_1D^{-1}+x_2D^{-2}\right]\cdot\frac{G_1(D)}{G_0(D)}|_t \tag{1}$$

Because of the code linearity we can do this operation for the both bits separately and sum the results

$$\left(x_1D^{-1}\cdot\frac{G_1(D)}{G_0(D)}\right)|_t+\left(x_2D^{-2}\cdot\frac{G_1(D)}{G_0(D)}\right)|_t \tag{2}$$

The generator matrix for the tellis section maps the initial state and the incoming bits to the output bits and to the next state value.

By merging the sections where the transition did not include any parity bit we have the generator matrix which first part is identity matrix. In other words the initial state and systematic bits are copied straight to the output. The transmitted coded bit in the trellis section can be viewed as parity bit calculated based on the previous state and the systematic bits whose encoded bits are punctured.

The code generator matrix for the section has the form $[\mathbf{I}\quad\mathbf{P}]$. Where **I** has the size of the memory length plus the amount of systematic bits in the merged trellis section. **P** describes the calculation of the parity bit and the value of the next state.

For example for our code with memory two (x_{m1} x_{m2}) and given puncturing we have 4 subsequent systematic bits (u_1 u_2 u_3 u_4). The trellis section is generated by multiplying

$$[x_{m2}\ x_{m1}\ u_1\ u_2\ u_3\ u_4\ ]\ \ [\mathbf{I}\quad\mathbf{P}] \tag{3}$$

The multiplication result with **P** gives us the parity bit and the next state value.

All possible values for the branch group are generated by selecting all possible input vector values in eq. 3. The dual code should be orthogonal to all the vectors in the branch group. The basis for such code can easily be calculated from the generator matrix for the branch group $[\mathbf{P}'\quad\mathbf{I}]$.

The values in the columns of **P** are one if the input vector element contributes to the particular output and zero otherwise.

The values for **P** can be found straightforwardly by proceeding trough the trellis. Above we described how the elements of **P** can be calculated by by using the multiplication with the generator polynomial.

The **P** contains two parts: the parity check calculation and calculation of the next state value. The state value is defined by the memory content of the encoder at the end of the trellis section. This content depends only on the denominator part of the generator polynome $\frac{1}{G_0(D)}$.

The content of the memory at the end of the section can be calculated by expressing the input systematic bits as a polynomial. Initiating the memory elements by the initial state in the trellis and diving the input polynomial by the denominator of the generator polynomial. This division gives us the bit sequence that is fed into the memory elements.

The content of the encoder memory with length m at some time moment t is defined by the output bit at division stage t and by last $m-1$ bits in the division sequence.

Because of the linearity of the code contribution of some particular bit to the memory content can be calculated separately for each input information bit (as we did in 2).

To each bit in the input polynomial corresponds one row in the generator matrix.

The values in the **P** for some particular bit are generated by dividing the elements in the input polynomial corresponding to this bit. The values to the **P** are selected at division stage We merge subsequent trellis stages where the transtion is described only by the t as last m division results.

For generating the values for the row in **P** corresponding to the parity bits one has to do similar division as for generating the contribution to the memory values. Now, however, one has to use the full generator polynomial and to select only the last value after the division.

We illustrate the implementation of our algorithm by an examplWe merge subsequenct trellis stages where the transtion is described only by the e. We puncture 1/3 rate turbo code to the 2/3 rate code with puncturing pattern [0 0 0 1] for one constituent encoder.

The initial trellis for the first code is given in figure a). The punctured bits are given in the brackets. The merged trellis for the section of the code is figure II b). In the figure we show the transitions only for the first state. Transition from state 00 to state 00 have many paralle paths in the initial trellis. In these paths in the merged trellis are shown only by one connecting edge and the parallel path patterns are given on that edge.

The represented trellis section contains only one encoded bit. The next state and the coded bit is identified based on the initial state and the incoming bits.

We calculated the generator matrix by evaluating the content of the feedback register and encoded bit at trellis stage $t=4$. (that corresponds to the trellis section shown in the figure).

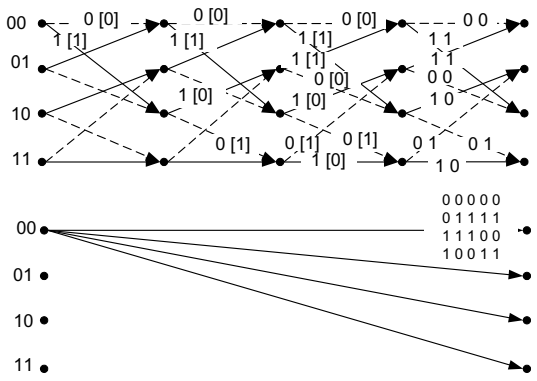

Fig. 2. The code trellis before a) and after merging b)

The generator matrix for the given code section is:

$$G(D) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

The first column selects the older bit in the encoder memory in initial stage. The second column is for the last bit in this memory. Columns 2 to 6 select the transmitted systematic bits. The seventh column describes the combination of input bits (and memory) used for evaluating the parity bits and eightth and ninth correspond for the memory content in the new stage.
The basis for the dual code for ths code section is

$$H(D) = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

All dual values for the branch group can be generated by the combinations of rows of the matrix $H(D)$.
The matrix $H(D)$ also is useful for generating trellis for the dual code. Its first two columns correspond to the initial state in the dual trellis. Columns $2-5$ tell us the bit values assigned to the edge of the trellis and the last two columns to which state this edge is connected.
For example the first row tells us that in the dual space the state [1 1] is connected to the state [0 1] and the edge is assigned value [1 0 1 1 0]
The total amount of codewords in this initial trellis section is $4 \cdot 2^4$ and from each state is 4 transitions to each other state. Total number of paths that one has to consider in normal trellis evaluation is $4 \cdot 2^4$.
The dimension of the dual space for this section is 3 and there are total 2^3 codewords. In the dual trellis there are four states and from each state there are only two outgoing transitions.

III. Decoding algorithm

We evaluate the loglikelihood of the received bit by using the dual code. The approach is described in [2]. Here we outline only the main properties of the algorithm.
We have a binary systematic codeword $\mathbf{c} = (c_1, c_2, ..., c_n)$ that is BPSK modulated and transmitted over the memoryless AWGN channel.
At the receiver we apply turbo decoding. In this purpose we calculate the loglikelihood of the bits (soft bits) at the output of each encoder.
In dual space soft bit for a binary systematic code is evaluated by the following formula [2]:

$$L(\hat{u}_k) = L_c(y_k) + L(u_k) + \ln \frac{\sum\limits_{\mathbf{c}' \in C^{\perp}_{k0}} \prod\limits_{j \in J_{c'}} \tanh\left(\frac{L(u_j,y_j)}{2}\right) + \sum\limits_{\mathbf{c}' \in C^{\perp}_{k1}} \prod\limits_{j \in J_{c'}} \tanh\left(\frac{L(u_j,y_j)}{2}\right)}{\sum\limits_{\mathbf{c}' \in C^{\perp}_{k0}} \prod\limits_{j \in J_{c'}} \tanh\left(\frac{L(u_j,y_j)}{2}\right) - \sum\limits_{\mathbf{c}' \in C^{\perp}_{k1}} \prod\limits_{j \in J_{c'}} \tanh\left(\frac{L(u_j,y_j)}{2}\right)} \tag{4}$$

Where u_k is modulator output for the input bit c_k. $y_k = u_k + n_k$ is the received signal after AWGN channel where n_k is the noise sample. $L_c(u_k)$ is the loglikelihood of the received channel output conditioned by the channel input. $L(u_k)$ is a-priori value of the bit. $L(u_j, y_j) = L_c(u_j) + L(u_j)$ if u_j is the information bit and $L(u_j, y_j) = L_c(u_j)$ if u_j is a parity check bit.
The set all dual codewors is denoted by $C^{\perp}$ and $C^{\perp}_{k0}$ is the subset where the bit $u_k = 0$ and in $C^{\perp}_{k1}$ it is accordingly $u_k = 1$. The bit k itself is excluded from the dualcodeword, i.e. when the multiplication of $\tanh$ is made.
As we see for finding the loglikelihood in the dual space we have to multiply the $\tanh\left(\frac{L(u_j,y_j)}{2}\right)$ of all the bits being 1 in the dual codeword and to exclude from this multiplication the term corresponding for k. That multiplication can be simplified by utilising the trellis structure of the dual code [4]. As for the normal MAP algorithm also for dual trellis we can devise an "forward" "backward" calculation based algorithm.
In forward calculation we gather into a state variable $A_i(s_i)$ all the dual codewords that will have common future and in "backward" calcualtion $B_i(s_i)$ contains the dual codewords with common beginning.
The can be described for each dual code trellis section

$$A_k(s_k) \prod_{j \in J_{c'_i}} \tanh\left(\frac{L(u_j,y_j)}{2}\right) B_{k+1}(s_{k-1}) \tag{5}$$

Where the set $J_{c'_i}$ describes the edge in a trellis section.
As we see the main difference compared to the MAP algorithm on the normal code trellis it that the path weight is calculated based on the multiplication of $\tanh\left(\frac{L(u_j,y_j)}{2}\right)$ of the bits being 1 in the dual code trellis edge.

IV. Simulation results

The purpose of the simulations is to investigate whether the performance of the decoding in dual space is comparable to the decoding in normal space. In simulations we use the code that was described in the example in the section 2.
The implementation of 4 in dual space corresponds to MAP algorithm in the normal space. Also as in the normal space we can simplify this algorithm.
We approximate the calculation of the multiplications of $\tanh$

on each edge by so called boxplus operation [3].

$$\prod_{j=1}^{J} \tanh(L(u_j)/2) \approx$$
$$\tanh\left(\frac{1}{2}\left(\prod_{j=1}^{J}(L(u_j)/2)\right)\cdot \min_{j=1,\ldots,J}|L(u_j)|\right)$$

We use this approximation only to evaluate the values on the edge.

We simulated the decoding in the dual space and compare it with the normal decoder. We normal decoder we applied both maxlogMAP and logMAP algorithm.

In the simulation we use packetw with 600 coded bits (400 information bits). We carried out 7 turbo decoding iterations.

The simulation results in figure 3 indicate that decoding in dual space and normal space have comparable performande. As expected the optimal logMAP decoder performs better than maxlogMAP based decoder and its equivalent boxplus using decoder in dual space.

Implementing the dual decoder based on evaluation of $\tanh$ at each stage does not operate well. Our guess is that it is because we run into numerical problems. The algorithm operates in normal domain not in logarithmic domain. In computations the dynamical range of the numbers is not sufficient.

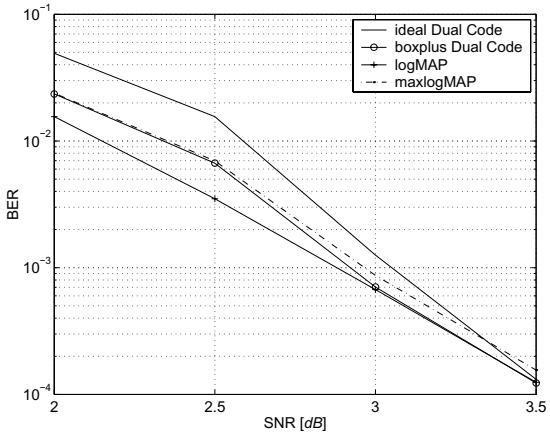

Fig. 3. BER for the dual and normal decoder

V. Conclusions

We show how punctured turbo code can easily be prepared for decoding in dual space. We show how to dualisize the section of the code and simulated the performance of the dual decoder.

As shown in the literature the dual space decoding of high rate codes is very efficient [5]. We show in this paper how to generate the trellis for the dual code of puntured recursive convolutional code.

The simulations show that in dual space the boxplus based algorithm has the same performance as the maxlogMAP algorithm in normal space.

References

[1] C. Hartmann, L. Rudolph, "An Optimum Symbol-by-Symbol Decoding Rule for Linear Codes," *IEEE Trans. Inform. Theory,* vol. IT-22, pp. 514-517, Sept. 1976.

[2] G. Battail, M.C. Decouverlaere, and P. Godlevski, "Replication decoding," *IEEE Trans. Inform. Theory,* vol. IT-25, pp. 332-345, May 1979.

[3] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory,* vol. 42, pp. 429-445, March 1996.

[4] J. Berkmann and C. Weiß, "On Dualzing Trellis-Based APP Decoding Algirithms," *Proc. 2002 IEEE Trans. Commun.,* vol.50, pp. 145, Nov. 2002.

[5] P.J. Schreier, D.J. Costello, "MAP decoding of linear block codes based on a sectionalized trellis of the dual code," *International Zurich Seminar on Broadband Communications,* pp. 271-278, Feb. 2000.