Multitree Decoding and Multitree-Aided LDPC Decoding

Maja Ostojic and Hans-Andrea Loeliger
Dept. of Information Technology and Electrical Engineering
ETH Zurich, Switzerland
Email: {ostojic,loeliger}@isi.ee.ethz.ch

***Abstract*—New decoding algorithms for linear codes are proposed. The first part of the paper considers decoding general binary linear codes by searching multiple trees, which is shown to achieve near maximum-likelihood performance for short block lengths.**

The second part of the paper considers decoding low-density parity check (ldpc) codes by means of repeated decoding attempts by standard sum-product message passing. Each decoding attempt starts from modified channel output, where some of the bits are clamped to a fixed value. The values of the fixed bits are obtained from multitree search.

I. Introduction

Tree search (sequential decoding) algorithms were traditionally used to decode convolutional codes [1], [2]. A related decoding method for short block codes was presented in [3]. In this paper, we propose, first, a new class of tree search decoding algorithms for linear codes, and second, the use of such algorithms to enhance message passing decoders of low density parity check codes. The main features of the proposed algorithms are the following:

- We use not only one tree, but several different trees, which enables several independent decoding attempts. The required structural information about the code may either be precomputed or generated on the fly.
- Each tree is explored such that only the scores of nodes at equal depth are compared. The maximum computational complexity is controlled by a free parameter.

The resulting algorithms achieve near maximum likelihood performance both for random linear codes and for low density parity check codes of short length. Similar performance has previously been reported in [4]–[6]. For a given performance, it seems that the multitree decoding algorithms proposed in this paper are somewhat faster (at the expense of more memory) than those of [4]–[6], but a fair comparison is not easy and outside of the scope of this paper.

In the second part of this paper, we focus on low-density parity check (ldpc) codes. We use a version of multitree search to provide multiple starting points for subsequent decoding attempts by standard iterative sum-product message passing.

The general idea of repeated sum-product decoding attempts using manipulated channel output was previously proposed in [7]–[9]. In contrast to this earlier work, our approach works also for longer ldpc codes.

In the simplest case, our multitree-aided ldpc decoding reduces to flipping and fixing a single bit at a number of

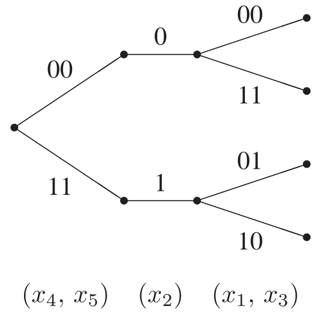

Figure 1. Code tree corresponding to (1).

different trial positions. Even this simple scheme significantly reduces the error probability. Additional gains are obtained by additional sum product decoding attempts from starting points obtained by nontrivial multitree search.

Throughout this paper, we will consider only binary linear codes and transmission over an additive white Gaussian noise (AWGN) channel with binary antipodal signaling.

The paper is structured as follows. Section II presents a basic tree search decoder which is used throughout this paper. Section III describes how to apply this tree search decoder to general linear codes using multiple trees. In Section IV we present a special version of multitree decoding for ldpc codes. A depth-limited version of the ldpc tree search decoder is then used in Section V to provide starting points for repeated decoding attempts by standard sum-product message passing.

II. A Tree Search Decoder

A. The Code Tree

A code tree is a graphical representation of the codewords of a code. To construct a code tree, we begin with a parity check matrix in row echelon form. In such a matrix, the number of leading zeros in row $m+1$ exceeds the number of leading zeros in row m. An example of such a parity check matrix is

$$H = \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left(\begin{array}{ccccc} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array} \right) \end{array}. \tag{1}$$

For every row h_i of $H = (h_1^T, \ldots, h_\ell^T)^T$, we define the incremental support set

$$S_\Delta(h_i) \triangleq (\text{support of } h_i) \setminus \bigcup_{j>i} S_\Delta(h_j). \tag{2}$$

For H as in (1), the incremental support sets are $(S_\Delta(h_3), S_\Delta(h_2), S_\Delta(h_1)) = (\{4,5\}, \{2\}, \{1,3\})$.

From such a parity check matrix H, we form a code tree as in Figure 1 as follows. Each section of the tree corresponds to a row of H. The sections in the tree from left to right correspond to the rows of H in the order $(h_\ell, h_{\ell-1}, \ldots)$. In each section, the branches of the tree are labeled with values for the bits in $S_\Delta(h)$ of the corresponding row h. The number of branches out of each node is $2^{|S_\Delta(h)|-1}$ for the corresponding row h. Each node defines a partial codeword determined by the branch labels on the path from the root node to the current node. Terminal nodes define a codeword.

B. The Search Algorithm

A code tree is explored as follows. We maintain a list of active nodes at each depth, which is initialized by the root node at depth zero and no nodes at larger depth. We use some score function which assigns a real value to each node. We will use different score functions for different classes of codes in the next sections. The higher the score of a node, the more promising the node appears. The score of a node at maximum depth is the log-likelihood of the codeword corresponding to the node.

We update the list of active nodes in a series of sweeps. In each sweep, we go sequentially through all depths $d = 1, 2, \ldots$ and perform the following steps at each depth d:

1) Select the active node at depth d with the highest score.
2) Generate its successor nodes at depth $d+1$ and insert them into the list of active nodes.
3) Delete the expanded node from the list of active nodes.

Note that in Step 2, in the special case where a node has only one (unique) extension to a codeword, then we generate this codeword right away.

Each sweep generates at least one new codeword. The search stops either when the score of a codeword exceeds a predefined threshold or after a maximum number of sweeps. Note that the list size at each depth can be limited to the maximum number of remaining sweeps.

III. Multitree Decoding of General Binary Linear Codes

In this section, we propose a new decoding algorithm for general binary linear codes using multiple code trees that are generated based on the received channel output. Each code tree is searched using the algorithm of Section II.

A. Reliability-Based Generation of Multiple Code Trees

For a given linear code of length n and a received channel output sequence $(y_1, \ldots, y_n)$ we form multiple code trees by repeating the following steps. Let H_0 be a (full rank) parity check matrix of the code.

1) Create a random permutation $(s_1, s_2, \ldots, s_n)$ of the index sequence $(1, 2, \ldots, n)$ by a procedure, to be described below, that favours high-reliability bits at the end.
2) Permute the columns of H_0 into the sequence $(s_1, s_2, \ldots, s_n)$. Call the result H'.
3) Bring H' into row echelon form by row operations. Call the result H.

The resulting matrix H defines a code tree as in Section II-A that favors high-reliability bits near the root of the tree.

For Step 1, we first assign to each index $\ell \in \{1, 2, \ldots, n\}$ the probability

$$p(\ell) = \gamma e^{|\alpha \log p(y_\ell|x_\ell=0)/p(y_\ell|x_\ell=1)|} \tag{3}$$

where x_ℓ denotes the channel input at time ℓ, where $\alpha \in \mathbb{R}$ is a free parameter (with $\alpha = 1$ by default), and where $\gamma \in \mathbb{R}$ is the scale factor required for $\sum_{\ell=1}^{n} p(\ell) = 1$. With these probabilities, we sequentially draw $s_n, s_{n-1}, \ldots, s_1$ from $\{1, 2, \ldots, n\}$ without replacement.

In the resulting code trees, nodes close to the root can have a large number of successor nodes; for a random (binary linear) code of dimension k, the size of the incremental support sets (from the bottom up) tends to be close to the sequence $k/2$, $k/4$, $k/8 \ldots$

B. Score Function

Given a received channel output sequence $y_1, \ldots, y_n$, the score of a node is the log-likelihood of the corresponding partial codeword, i.e., $\sum_\ell \log p(y_\ell|x_\ell)$, where x_ℓ are the bits in the partial codeword.

C. Multitree Search

We try multiple independent decoding attempts, each with a different code tree, and stop if a codeword is accepted or if the maximum number of decoding attempts is reached. A codeword is accepted if its log-likelihood exceeds some acceptance threshold. The threshold is determined experimentally such that the simulated error rates are not significantly affected. Such an acceptance threshold greatly reduces the average decoding complexity. The worst-case complexity is limited by the maximum number of sweeps per tree and the maximum number of trees.

D. Simulation Results

We use an AWGN channel with $\{-1, +1\}$-signaling. The signal-to-noise ratio is $10\log_{10}(1/\sigma^2)$, where σ^2 is the variance of the noise.

Simulation results for the proposed decoding algorithm are shown in Figures 2 and 3. In both figures, codes of rate 1/2 and length $n = 100$ are used. Also shown in the figures is a lower bound on the word error rate of the maximum-likelihood (ML) decoder. This lower bound is obtained as follows. Whenever the multitree search decoder finds a codeword whose log-likelihood exceeds the log-likelihood of the transmitted codeword, a word error is counted for the ML decoder.

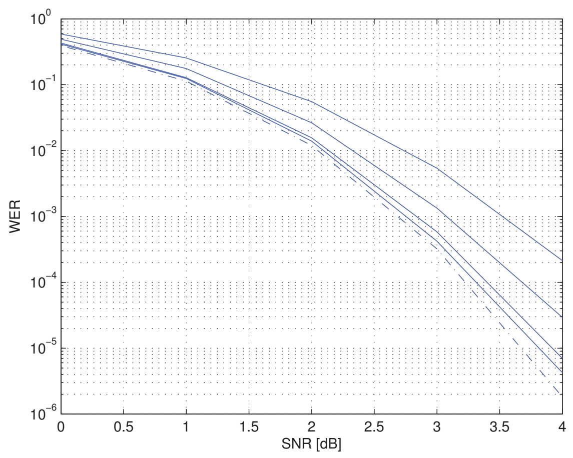

Figure 2. Simulated word error rates for multitree decoding of a random rate-1/2 code of length $n = 100$. From top to bottom: (i) max 200 sweeps in a single tree; (ii) max 1000 sweeps in a single tree; (iii) max 100 sweeps in each of 10 different trees; (iv) max 200 sweeps in each of 10 different trees; (v) ML lower bound.

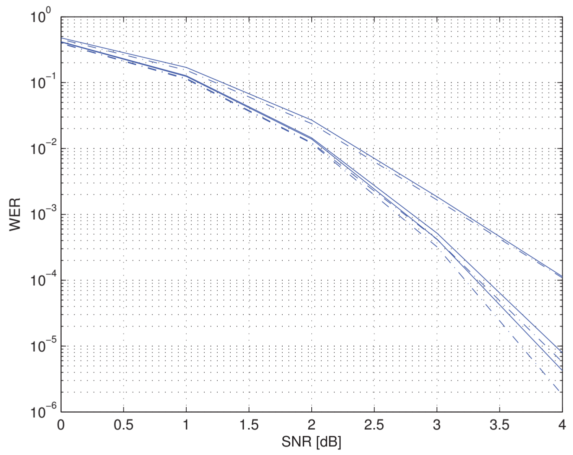

Figure 3. Simulated word error rates for multitree decoding (solid lines) and corresponding ML lower bounds (dashed lines) for three different rate-1/2 codes of length $n = 100$. Top: (3,6) ldpc code. Middle: (5,10) ldpc code. Bottom: random code (same as in Fig. 2). All codes are decoded by max 200 sweeps in each of 10 different trees.

IV. Multitree Decoding of LDPC Codes

In this section, we describe a version of multitree decoding as in Section III that is tailored to ldpc codes. We construct code trees such that nodes close to the root have a much smaller number of successors than the code trees in Section III. In addition, we will use precomputed code trees and a more elaborate score function.

A. Code Tree from a Low-Density Parity Check Matrix

Assume that we are given a (full rank) low-density parity check matrix H_0 with n columns and $n - k$ rows. We will describe a procedure to obtain an equivalent parity check matrix H in row echelon form that preserves some of the sparseness of H_0. Due to the sparseness of H, the code tree has much fewer branches out of nodes near the root than the code trees of Section III-A. The matrix H is then used to form a code tree as in Section II-A.

A greedy probabilistic method to transform H_0 into H works as follows:

1) Bring the last m rows into row echelon form by row and column permutations as explained below.
2) Bring the first $n - k - m$ rows into row echelon form by row operations.

Note that the last m rows retain the sparseness of H_0. An example for $m = n - k - 2$ is

$$H = \left(\begin{array}{ccccccccccc} 1 & \dots & & & & & & & & & \\ 0 & 1 & \dots & & & & & & & & \\ \hline 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \dots & 0 \\ \vdots & & & & & & & & & & \vdots \\ 0 & & & & \dots & & & & 0 & 1 & 1 \end{array}\right). \quad (4)$$

This method can fail if m is chosen too large. In that case, we decrease m by one and try again. For a regular $(3, 6)$ ldpc code, a good choice for (the initial value of) m is $3/5$ of the number of rows.

The details of Step 1 are as follows. We build the last m rows of H row by row starting with the bottom row. The i-th row is constructed as follows:

1.1) Swap the row h_i with some row from $h_1, \dots, h_i$, such that after the permutation the incremental support $S_\Delta(h_i)$ (2) is minimal and h_i intersects with at least one of the rows $h_{i+1}, \dots h_{n-k}$.
1.2) Permute the columns of h_i in $S_\Delta(h_i)$ such that the nonzero entries appear immediately to the left of the leftmost nonzero entry in the row below (i.e., h_{i+1}).

For an example (with $m = n - k$), consider

$$H_0 = \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left(\begin{array}{ccccc} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array}\right) \end{array} \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \quad (5)$$

In step $1.1)$ we choose row 1 for the last row. After step $1.2)$ we obtain

$$H_1 = \begin{array}{c} \begin{array}{ccccc} 2 & 4 & 5 & 1 & 3 \end{array} \\ \left(\begin{array}{ccccc} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right) \end{array}. \begin{array}{c} \\ 3 \\ 2 \\ 1 \end{array} \quad (6)$$

For the second row, we can choose either row 2 or 3 (as indexed before the permutation). We arbitrarily choose row 2 and obtain the final result

$$H = \begin{array}{c} \begin{array}{ccccc} 5 & 4 & 2 & 1 & 3 \end{array} \\ \left(\begin{array}{ccccc} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right) \end{array}. \begin{array}{c} \\ 3 \\ 2 \\ 1 \end{array} \quad (7)$$

B. Reliability-Based Selection of Precomputed Trees

We use precomputed code trees, which are obtained by random choices among the eligible rows h in Step 1.1 in Section IV-A. (We only use trees with $|S_\Delta(h_{n-k-m})| \leq 3$, in order to restrict the number of branches out of the corresponding node in each tree.)

We precompute a large number of trees, so that, for a given received channel output sequence $y_1, \ldots, y_n$, we can find some trees in which high reliability bits appear close to the root. In particular, we choose for decoding a fixed number of trees with high value of $\sum_{\ell \in I} \log p(y_\ell|\tilde{x}_\ell)$ where $\tilde{x}_\ell$ is the hard decision for bit ℓ and I indicates the first bits in the tree. The size $|I|$ is a free parameter.

C. Score Function

The score of a node with partial codeword $x_I = v_I$ is

$$\max_{x \in C' : x_I = v_I} \log p(y|x), \tag{8}$$

where C' is a relaxation of the original code C and $I \subset \{1, \ldots, n\}$ denotes the bits in the partial codeword. The score (8) is an upper bound on the log-likelihood of any codeword $c \in C$ which passes through this node.

We obtain C' by choosing a parity check matrix H' that consists of a subset of the rows in H. We choose the rows h' in H' to satisfy

- support$(h') \cap I \neq \emptyset$
- (support$(h'_i) \setminus I) \cap ($support$(h'_j) \setminus I) = \emptyset$ for $i \neq j$.
- h' is unsatisfied given $x_I = v_I$ and the hard decisions of the bits not in I.
- $s(h') \setminus I$ contains only high-reliability bits.

D. Simulation Results

We use the same multitree procedure as in Section III-C with an acceptance threshold. Some simulation results with this decoding algorithm are shown in Fig. 4. These results are obtained with 500 precomputed trees, out of which we pick at most 5 trees (by comparing first 40 bits in the tree) for decoding.

V. Multitree-Aided LDPC Decoding

We now use the multitree search algorithm to help a standard sum-product message passing decoder when it fails to find a codeword. We repeatedly select a subset of bits and use a tree search to find the most likely decisions for these bits (or a list thereof). We then replace the channel information of the selected bits with the tree search output decisions and restart the message passing decoder. We continue until the decoder finds a codeword (which we verify with the parity check equations) or a maximum number of trials is reached.

A. Single Bit Flip-Aided Decoding

In the simplest possible case we fix the value of only one bit. The tree search is trivial for this problem and the task reduces to identifying suitable bit positions. We are specifically looking for a bit with high reliability, whose hard decision is likely to be wrong. We then restart the message passing decoder with the value of the chosen bit forced to the flipped hard decision.

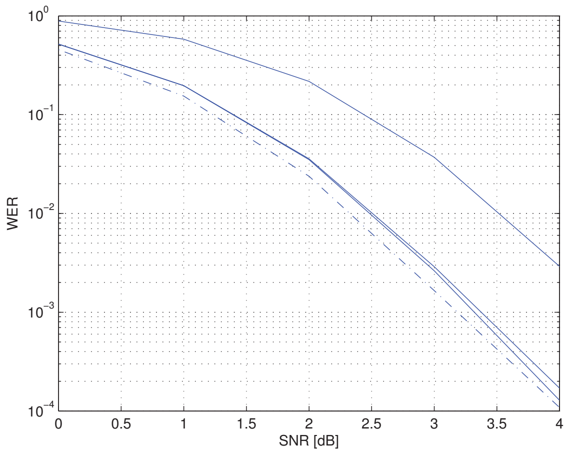

Figure 4. Simulated word error rate (solid lines) for different decoders of a regular (3,6) ldpc code of length $n = 100$ (same as in Fig. 3) and ML lower bound (dashed line). Top: standard sum-product message passing. Middle: multitree decoding as in Section III. Bottom: multitree decoding as in Section IV. Both multitree decoders use max 50 sweeps in 5 different trees.

We identify candidate bits as follows. For each unsatisfied parity check equation h, we compute the log-likelihood $\sum_\ell \log p(y_\ell|x'_\ell)$, where ℓ goes over the support of this parity check and x' maximizes the log-likelihood such that h is satisfied. We order the parity checks according to this log-likelihood. Beginning with the parity check with highest log-likelihood, we go through the list of checks and choose the two least reliable bits in every check until the maximum number of bits is reached.

B. Multitree-Aided Decoding

We use a depth-limited version of the multitree search algorithm presented in Section IV. We fix a value $m < n - k$ and do the permutations as in Step 1) on the parity check matrix. We then construct a tree up to depth m from this matrix. We decode this depth-limited tree with the score function presented in Section IV-C.

We generate multiple trees by using a different bottom row as a starting point for the permutations of Step 1) in Section IV-A. We don't precompute any trees but use the received sequence $y_1, \ldots, y_n$ to break ties in Step 1.1) when choosing the next row for permutation. We base the tie-breaking rule on the following observations:

- A satisfied parity check equation with high reliability bits is likely to contain no error.
- An unsatisfied parity check equation with high reliability bits is likely to contain exactly one error.

For the bottom $m/2$ rows, we choose h such that h is not satisfied and the bits constrained by h have high reliability. For the next $m/2$ rows we choose h which has high reliability bits and is satisfied. With this row selection method we aim to choose a subset of bits which contains some high reliability flips. The satisfied high reliability parity check rows are added to help the tree search decoder find the correct decisions.

C. Simulation Results

Some simulation results for this decoding algorithm are shown in Figures 5 and 6. Both codes are almost-regular (3,6) codes (obtained via progressive edge growth) taken from [10]. For every received sequence, we try sum-product message passing decoding, then single bit flip-aided message passing decoding, and if this fails, too, multitree-aided decoding with trees of maximum depth $m = 10$. We use 10 sweeps in each tree search with no threshold and hand a list of 10 partial codewords to the message passing decoder. In both figures, the very simple bit flipping scheme performs surprisingly well.

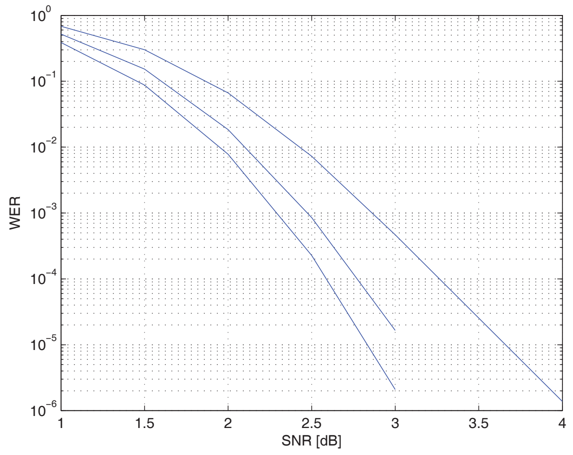

Figure 5. Simulated word error rates for a $(3, 6)$ low-density parity check code of length $n = 504$. Top: standard sum-product message passing (50 iterations). Middle: repeated sum-product decoding attempts starting from a single forced bit flip in up to 100 trial positions. Bottom: max 10×100 additional sum-product decoding attempts starting from partial codewords (involving 10 parity checks) obtained from multitree search using 100 different trees.

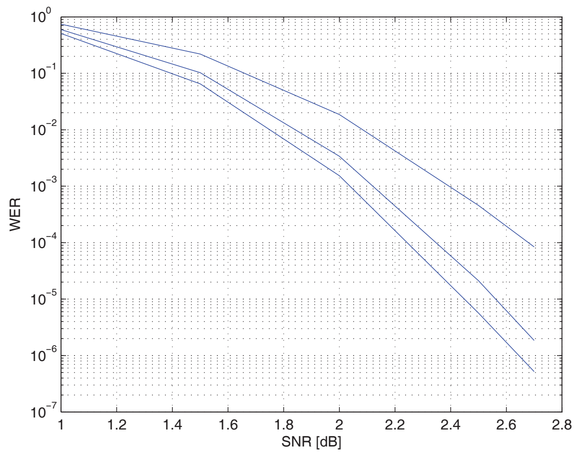

Figure 6. Simulated word error rates for a $(3, 6)$ low-density parity check code of length $n = 1008$. Top: standard sum-product message passing (50 iterations). Middle: repeated sum-product decoding attempts starting from a single forced bit flip in up to 200 trial positions. Bottom: max 10×200 additional sum-product decoding attempts starting from partial codewords (involving 10 parity checks) obtained from multitree search using 200 different trees.

VI. Conclusions

We have proposed new multitree decoding algorithms for general binary linear block codes, which are shown to achieve near-maximum-likelihood performance for short block length. We have also proposed variations of this approach for ldpc codes. In particular, multitree search can be used to provide starting points for repeated decoding attempts by standard sum-product message passing. This last approach appears to be particularly attractive in its simplest form, where the multitree search reduces to flipping and fixing a single bit at a number of different trial positions. A more detailed study of multitree search decoding can be found in [11].

References

[1] J. M. Wozencraft, "Sequential decoding for reliable communication," *IRE Conv. Rec.,* vol. 5, pt. 2, pp. 11–22, 1957.

[2] R. Johannesson and K. Sh. Zigangirov, *Fundamentals of Convolutional Codes.* Piscataway: IEEE Press, 1999.

[3] Y. S. Han, C. R. P. Hartmann, and C.-C. Chen, "Efficient priority-first search maximum-likelihood soft-decision decoding of linear block codes," *IEEE Trans. on Inform. Theory*, vol. 39, no. 5, pp. 1514–1523, Sept. 1993.

[4] M. P. C. Fossorier and Shu Lin, "Computationally efficient soft-decision decoding of linear block codes based on ordered statistics," *IEEE Trans. Inform. Theory,* vol. 42, no. 3, pp. 738–750, May 1996.

[5] A. Kothiyal, O. Takeshita, Wenyi Jin, and Marc Fossorier, "Iterative reliability-based decoding of linear block codes with adaptive belief propagation," *IEEE Commun. Let.,* vol. 9, no. 12, pp. 1067–1069, Dec. 2005.

[6] Sheng Tong, D. Lin, A. Kavčić, Li Ping, and B. Bai, "On the performance of short forward error-correcting codes," *IEEE Commun. Let.,* vol. 11, no. 11, Nov. 2007.

[7] H. Pishro-Nik and F. Fekri, "Improved decoding algorithms for low-density parity-check codes," Proc. of the 3rd Int. Conf. on Turbo Codes and Relat. Topics, Brest, France, Aug. 2003.

[8] N. Varnica, M. P. C.Fossorier, and A. Kavčić, "Augmented belief propagation decoding of low-density parity check codes," *IEEE Trans. Commun.*, vol. 55, no. 7, pp. 1308–1317, July 2007.

[9] M. Zhou and Xiqi Gao, "Improved augmented belief propagation decoding based on oscillation," 14th Asia-Pacific Conf. on Comm., Tokyo, 2008.

[10] D. J. C. MacKay, Encyclopedia of Sparse Graph Codes, http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

[11] M. Ostojic, *Multitree Search Decoding of Linear Codes*, Ph.D. thesis, Swiss Federal Institute of Technology, 2010.