

An Ordered-Statistics-Based Approximation of the Sum-Product Algorithm

Xiao-Yu Hu and Thomas Mittelholzer

IBM Research, Zurich Research Laboratory, Rüschlikon, Switzerland

Abstract— A complexity- and delay-efficient simplification of the sum-product algorithm (SPA) for decoding low-density parity-check (LDPC) codes is presented. The key feature of the new algorithm consists of a modification of the complexity-intensive and delay-causing update equations at the check nodes of the factor graph of the LDPC code. The modified update equations at a check node are based on ordering the reliability values of the incoming messages and on using a balanced tree topology to achieve optimum parallel processing. Furthermore, the complexity of the new algorithm can be adjusted: the least complex version of the algorithm corresponds to the so-called min-sum approximation, and the most complex version gives the full SPA.

Keywords— sum-product algorithm, min-sum algorithm, low-density parity-check codes.

I. INTRODUCTION

OWING to their outstanding performance, low-density parity-check (LDPC) codes [1], are currently recognized to be the best class of codes to approach the Shannon limit efficiently, using iterative decoding [2]. Efficient implementation of the decoding algorithm in hardware has become an area of increased interest (see e.g. [3, 4] and references therein). In particular, substantial reductions of the total decoding complexity have been obtained by considering simplifications of the core operations in the sum-product algorithm (SPA).

Here we focus on various binary tree representations associated with the check nodes in the factor graph of an LDPC code. We propose a modification of the highly complex and delay-causing update equations at the check nodes that is based on ordering the reliability values of the incoming messages. Specifically, these messages are split into the set of a few least reliable messages and the set of all other messages, which are treated as being fully reliable. In this way, the complexity of the update equations can essentially be reduced to the case of only a few incoming messages. This reduction in complexity is most pronounced for high-rate LDPC codes, where each check node is connected to a large number of symbol nodes. The proposed simplification does not rely on reduced-complexity approximations of the core operations; therefore, one can achieve additional complexity reductions by applying such approximate core operations [3-5].

A critical performance issue of all turbo-like codes is the decoding delay that is inherent in iterative decoding of block codes. Following [4, 6], we devote special attention

to low-delay implementations, which are based on balanced tree topologies for maximum parallel processing.

The paper is organized as follows. In Section 2, the SPA is reviewed, and the complexity of the different parts discussed briefly. In Section 3, different implementations for the check-node updates are considered in terms of special factor graphs that are binary trees. Section 4 is devoted to the use of ordered statistics of the incoming messages at the check nodes and to simplifications of the factor graph, both of which result in a reduction of the delay as well as of the total complexity. In Section 5, simulation results comparing the full SPA with the proposed simplified versions are shown. Finally, in Section 6, the key features of the proposed simplified algorithm are summarized.

II. THE SUM-PRODUCT ALGORITHM IN THE LOG-LIKELIHOOD DOMAIN

The SPA is an efficient iterative algorithm for decoding LDPC codes. In particular, given a received word $\mathbf{y} = [y_1, y_2, \dots, y_N]$, which corresponds to some transmitted binary codeword $[x_1, x_2, \dots, x_N]$, the SPA updates in each iteration step its “temporal beliefs” about the components x_n of the transmitted codeword given \mathbf{y} , and, eventually, makes a decision as to which codeword components were sent.

We describe the SPA in the log-likelihood domain using similar notations as in [7]. For a given binary sparse parity-check matrix H , we denote by $\mathcal{M}(n)$ the set of check nodes that are connected to symbol nodes n , i.e., the set of indices that are “1” in the n -th column of the parity-check matrix H . Similarly, we denote by $\mathcal{L}(m)$ the set of symbol nodes that participate in the m -th parity-check equation, i.e., the set of indices that are “1” in the m -th row of H . We denote by $\mathcal{L}(m) \setminus n$ the set-theoretic difference of $\mathcal{L}(m)$, from which the n -th symbol node is excluded. Similarly, for the check-node sets, we introduce the set-theoretic difference $\mathcal{M}(n) \setminus m$. We denote by $L(q_{n \rightarrow m})$ the message that symbol node n sends to check node m , which corresponds to the log-likelihood ratio (LLR) of some temporal likelihood that the n -th symbol is “0” or “1”. Similarly, $L(r_{m \rightarrow n})$ denotes the message that the m -th check node sends to the n -th symbol, which again is an LLR expressing the temporal belief that the m -th check node has about the n -th symbol being “0” or “1”. The SPA that is based on log-likelihood messages can be stated as follows.

Xiao-Yu Hu and Thomas Mittelholzer are with IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland. E-mails: {xhu, tmi}@zurich.ibm.com.

1. Initialization

Each symbol node n is assigned an initial LLR $L(p_n)$. In the case of equiprobable inputs on a memoryless AWGN channel,

$$L(p_n) = \frac{2}{\sigma^2} y_n,$$

where the σ^2 is the noise variance, and y_n the corrupted value of input symbol x_n . For every incidence pair (m, n) , i.e., pair of indices satisfying $H_{m,n} = 1$, the messages $L(q_{n \rightarrow m})$ and $L(r_{m \rightarrow n})$ are initialized as follows:

$$\begin{aligned} L(q_{n \rightarrow m}) &= L(p_n), \\ L(r_{m \rightarrow n}) &= 0. \end{aligned}$$

2. Check-node update

Each check node m gathers the incoming messages $L(q_{n \rightarrow m})$, and updates the belief on symbol n based on the information from all other bits connected to check node m :

$$L(r_{m \rightarrow n}) = 2 \tanh^{-1} \left\{ \prod_{n' \in \mathcal{L}(m) \setminus n} \tanh[L(q_{n' \rightarrow m}/2)] \right\}. \quad (1)$$

The outgoing message $L(r_{m \rightarrow n})$ is referred to as extrinsic information because it does not depend on the incoming message $L(q_{n \rightarrow m})$.

3. Symbol-node update

Each symbol node n propagates its likelihood information to all the check nodes that connect to it. The outgoing messages are calculated as

$$L(q_{n \rightarrow m}) = L(p_n) + \sum_{m' \in \mathcal{M}(n) \setminus m} L(r_{m' \rightarrow n}).$$

4. Decision

The decoder obtains the total temporal *a-posteriori* information for symbol n by summing the likelihoods from all the check nodes that connect to bit n :

$$\lambda_n = L(p_n) + \sum_{m \in \mathcal{M}(n)} L(r_{m \rightarrow n}).$$

The algorithm iterates until a valid codeword has been found, i.e., the hard decision of the temporal *a-posteriori* vector $[\lambda_1, \dots, \lambda_N]$ satisfies the parity-check matrix H , or a preset maximum number of iterations has been reached.

Obtaining an efficient implementation of the symbol-node updates is obvious. By forming first the *a-posteriori* information λ_n , the extrinsic information terms are given by $L(q_{n \rightarrow m}) = \lambda_n - L(r_{m \rightarrow n})$. Thus, the total computational load for a symbol-node update is only $2|\mathcal{M}(n)|$ additions, where the cardinality $|\mathcal{M}(n)|$ denotes the degree of the variable node n .

The check-node update is the most complex part of the SPA. Two issues influence its computational complexity:

the topology of the multiple outgoing messages and the implementation of the core operation. Consider a regular LDPC code of rate $R \geq 1 - j/k$, where j and k are the column and the row weight of its parity-check matrix, respectively. If this code is of high rate, then it consists of check nodes that are connected to many variables. For instance, the check nodes of LDPC codes with $R \geq 0.9$ and a column weight $j = 4$ are connected to more than 40 variables, i.e., $k > 40$. For an irregular LDPC code, the largest number of row weights can be even larger.

The core operation of the check-node update in (1) is the hyperbolic tangent function, which apparently seems difficult to implement in digital hardware. Analog realizations of the hyperbolic tangent function have been investigated in [8, 9], with the aim of facilitating extremely high-speed applications. The main difficulty in analog circuits is not the complexity of the hyperbolic tangent function but rather stability and synchronization issues. In Gallager's approach [1], the core operation of the check-node update in (1) was transformed into another form, i.e.,

$$2 \tanh^{-1} (\tanh(\lambda/2) \tanh(\mu/2)) = \text{sign} \lambda \text{sign} \mu \cdot f(f(|\lambda|) + f(|\mu|)),$$

where

$$f(x) = \ln \frac{e^x + 1}{e^x - 1}$$

is an involution for $x > 0$. By introducing the involution $f(x)$, one can run check-node updates efficiently from both the computational- and time-complexity viewpoints in a way similar to the efficient implementation of symbol-node updates described above. However, it is not evident how to implement the involution function $f(x)$ efficiently in either analog or digital circuits because of its singularity at 0.

III. BINARY TREES FOR THE CHECK-NODE CONSTRAINTS

The motivation to consider binary trees is based on the fact that the complexity-intensive computation of the check-node update can be obtained by repeated application of the identity

$$2 \tanh^{-1} (\tanh(\lambda/2) \tanh(\mu/2)) = \log \frac{1 + e^{\lambda+\mu}}{e^\lambda + e^\mu}. \quad (2)$$

Moreover, as noted in [10], the right-hand side in (2) is the LLR of the binary sum of two statistically independent binary random variables U and V , where $\lambda = L(U)$ and $\mu = L(V)$ are their LLRs, i.e.,

$$L(U \oplus V) = \log \frac{1 + e^{L(U)+L(V)}}{e^{L(U)} + e^{L(V)}}. \quad (3)$$

For the right-hand side in (3), which is a function of $L(U)$ and $L(V)$, we will use the well-established notation $L(U) \boxplus L(V)$ [10]. The domain of the \boxplus -operation can be extended to the set of real numbers together with $\pm\infty$. In this way, one obtains a monoid with ∞ as neutral element [10]. With

this notation, the check-node update (1) can be rewritten as

$$L(r_{m \rightarrow n}) = \boxplus_{n' \in \mathcal{L}(m) \setminus n} L(q_{n' \rightarrow m}/2), \quad (4)$$

where $\boxplus_{n'=1 \dots k} L(U_{n'})$ denotes the k -fold \boxplus -sum.

Without loss of essential generality, we will focus on a single check node, say $m = 1$, and assume that it checks the binary input symbols x_1, x_2, \dots, x_k . The factor graph for the check equation $x_1 \oplus \dots \oplus x_k = 0$ is a tree with k leaves, in which the root corresponds to check node $m = 1$ and the k symbols represent the leaves. To reduce the check-node updates (1), operating on $k - 1$ arguments, to a repeated application of the 2-argument formula (3), we transform the k -ary tree into a factor graph, which is a binary tree, by using suitable state variables S_ν . Two trees that achieve this task are shown in Fig. 1.

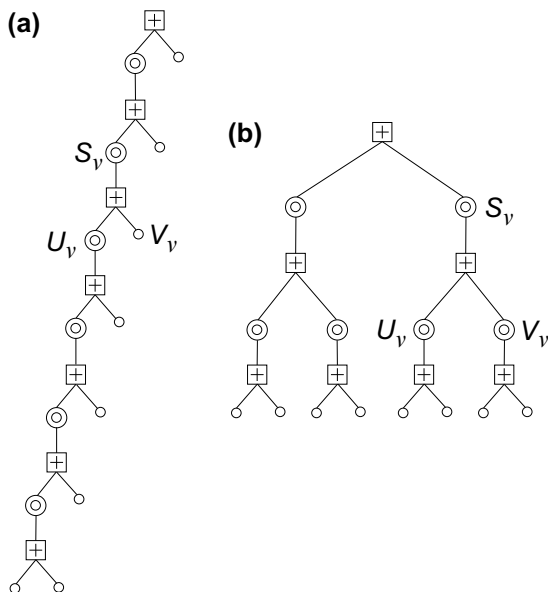


Fig. 1. Two factor graphs representing the single parity check $x_1 \oplus \dots \oplus x_k = 0$ (for $k = 8$): (a) a maximally unbalanced and (b) a balanced tree.

In Fig. 1, the double circles represent binary state variables. Each state variable, say S_ν , is the modulo-2 sum of the two symbol or state nodes, say U_ν and V_ν , checked by the binary check node leading towards state node S_ν (starting from the leaves towards the root), i.e., $S_\nu = U_\nu \oplus V_\nu$. The check-node updates (1) can be computed by making a forward and a backward pass on either of these factor graphs using the core function (3) or some approximation thereof [3-5]. For both factor graphs in Fig. 1, the total computational load for check node $m = 1$ consists of the forward recursive computation of $L(S_\nu)$ for the $k - 2$ state variables, the backward recursion for the latter, and the final backward recursion step to the leaves, which amounts to $3(k - 2)$ core operations $L(U) \boxplus L(V)$ (see [11] for the complexity computation on the maximally unbalanced tree).

The factor graph in Fig. 1(a) will be referred to as the maximally unbalanced tree (note that this factor graph is topologically equivalent to a binary tree obtained by deleting all state variables). Various simplifications of the SPA

have been derived from this maximally unbalanced tree [3, 5]. The other factor graph, Fig. 1(b), which is a balanced tree, has been proposed in [6] and implicitly in [4]. For the design of parallel algorithms, the balanced tree clearly results in a much smaller delay, i.e. about $2 \log k$ core operations, compared with a delay of about k consecutive core operations for the maximally unbalanced tree (when running the forward and backward passes in parallel).

In this paper, we propose an algorithm that operates on a strongly reduced factor graph for each single parity-check equation. This simplified balanced factor graph is obtained by searching for the z ($z \geq 2$) least reliable symbol nodes in each parity-check equation.

IV. ORDERED STATISTICS FOR CHECK-NODE UPDATES

This section is devoted to simplifying the balanced tree factor graph by using the ordered statistics of the reliability values of the symbol nodes. An example of such a simplification is given by a parallel min-sum version of the SPA as described in the following subsection.

A. Parallel Min-Sum Check-Node Updates

We consider the SPA on the balanced tree with a simplified core function based on the well-known minimum approximation [10]

$$L(U) \boxplus L(V) \approx \text{sign}(L(U))\text{sign}(L(V)) \cdot \min\{|L(U)|, |L(V)|\}. \quad (5)$$

This simplified SPA is known as the ‘‘min-sum algorithm’’ [6]. By performing the min-sum update rule on the factor graph for the check equation $x_1 \oplus \dots \oplus x_k = 0$ [see Fig. 1(b)], one obtains approximate values for the extrinsic LLRs $L(r_{1 \rightarrow n})$, $n = 1, \dots, k$, which are given by

$$L(r_{1 \rightarrow n}) \approx \begin{cases} \text{sign}[L(S)]\text{sign}[L(u_n)]L_m & \text{if } L_m < |L(u_n)| \\ \text{sign}[L(S)]\text{sign}[L(u_n)]L_{m'} & \text{otherwise.} \end{cases} \quad (6)$$

Here $S = x_1 \oplus \dots \oplus x_k$, and L_m and $L_{m'}$ denote the LLRs of the least and second least reliable symbol node, respectively. In particular, L_m and $L_{m'}$ are obtained by partially sorting the reliability magnitudes $|L(u_1)|, |L(u_2)|, \dots, |L(u_k)|$ to obtain the two symbol nodes with the two smallest values.

The approximations (6) can be computed directly from $\text{sign}[L(S)]$, which is obtained from the forward pass, and no backward recursion is needed, provided that the LLRs L_m and $L_{m'}$ of the two least reliable symbol nodes are known. The computation of L_m and $L_{m'}$ is a partial-sorting problem, which can be done efficiently in $\log_2(k)$ time. Note that $\text{sign}[L(S)]$ is obtained by a simplified forward pass, which requires only sign computations. Furthermore, this forward pass and the partial sorting can be done in parallel on two trees of depth $\log_2(k)$. Hence, the min-sum algorithm for the check-node updates runs in time $\log_2(k)$,

where the core operations are either comparisons or multiplications of two signs. The total number of operations required is about $k \log_2(k)$ comparisons and k sign operations.

B. Sorting-Based Check-Node Updates

By carrying the sorting idea of the preceding subsection further, we will obtain a family of reduced-complexity algorithms that approximate the check-node updates in the SPA. The balanced tree for the forward recursion can also be considered as a diagram for the sorting algorithm often referred to as merge-sorting [12] that is based on merging two ordered lists. Starting with the magnitudes $|L(x_1)|, |L(x_2)|, \dots, |L(x_k)|$ at the leaves of the tree, these k nonnegative values can be ordered within a time complexity of about $\log_2(k)$ comparison operations. The sorting algorithm determines a permutation

$$\pi = \begin{pmatrix} 1 & 2 & \dots & k \\ i_1 & i_2 & \dots & i_k \end{pmatrix} \quad (7)$$

of the k symbols such that the permuted reliability values are ordered, i.e., $|L(x_{i_1})| \leq |L(x_{i_2})| \leq \dots \leq |L(x_{i_k})|$.

The key idea of the new algorithm is to select a fixed number, say z , of least reliable symbol nodes x_{i_1}, \dots, x_{i_z} , and to treat all remaining symbol nodes as being fully reliable, i.e.,

$$|L(x_{i_{z+1}})| = |L(x_{i_{z+2}})| = \dots = |L(x_{i_k})| = \infty. \quad (8)$$

This amounts to keeping the original soft values for the z least reliable symbol nodes and approximating the original soft values for the other $k - z$ symbol nodes by the hard decisions $\pm\infty$.

The distinction of symbol nodes with soft and hard decisions can be reflected in the construction of a partially balanced binary factor graph for the updates of the check node corresponding to $x_1 \oplus \dots \oplus x_k = 0$ (see Fig. 2). This partially balanced binary tree, which we will call a

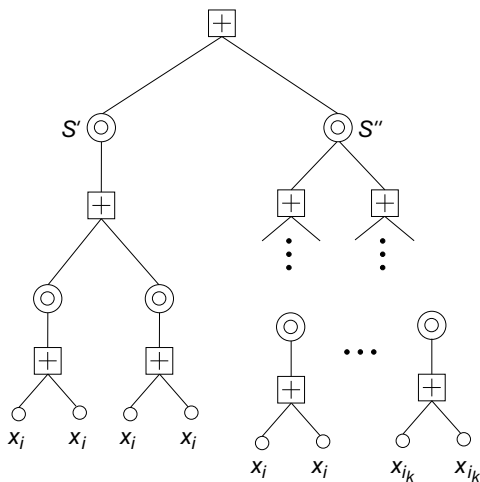


Fig. 2. Reliability-based balanced tree separating the $z = 4$ least reliable and the $k - z$ most reliable symbol nodes in parity check $x_1 \oplus \dots \oplus x_k = 0$.

reliability-based balanced tree, is obtained as follows. From the soft value symbols x_{i_1}, \dots, x_{i_z} , we form a balanced binary sub-tree with the root given by the binary state variable $S' = x_{i_1} \oplus \dots \oplus x_{i_z}$. Similarly, a balanced binary sub-tree with the root $S'' = x_{i_{z+1}} \oplus \dots \oplus x_{i_k}$ is formed from the hard-decision symbol nodes $x_{i_{z+1}}, \dots, x_{i_k}$. Eventually, the two sub-trees are connected via a binary parity-check node, which becomes the root of the reliability-based balanced tree.

The complexity of the SPA on the partially balanced factor graph is mainly determined by the complexity on the left sub-tree emanating from state node S' because full soft reliability values are computed only on this sub-tree. On the right sub-tree, which emanates from state S'' , the forward pass of the SPA needs only sign computations because all leaves have LLRs that are $\pm\infty$ (note that ∞ is the neutral element with respect to the \boxplus -operation).

To obtain a low-latency implementation of the check-node updates, we simplify the partially balanced tree of Fig. 2 further by keeping only the left sub-tree with the root S' . The binary value of the symbol node at the root, $S' = \sigma$, is determined by the hard-decision symbols, i.e.,

$$\begin{aligned} (-1)^\sigma &= \prod_{n=z+1}^k \text{sign}(L(x_{i_n})) \\ &= \text{sign}(L(S)) \prod_{n=1}^z \text{sign}(L(x_{i_n})). \end{aligned} \quad (9)$$

The resulting factor graph is shown in Fig. 3. Note that this graph can be further simplified by removing the state node S' and by incorporating the constraint $S' = \sigma$ into the check node at the root. The SPA on this simplified factor graph can either be carried out in its full version or by using some simplification for the \boxplus -operations (see e.g. [4]).

For $z = 2$, the algorithm essentially corresponds to the min-sum approximation. However, there is the following slight difference: in the min-sum algorithm the reliability of the messages for the hard-decision nodes are approximated by the minimum $|L(x_{i_1})|$ in contrast to the approximation $|L(x_{i_1})| \boxplus |L(x_{i_2})|$ in the new algorithm.

For high-rate LDPC codes, the sorting-based check-node updates provide substantial savings in complexity: com-

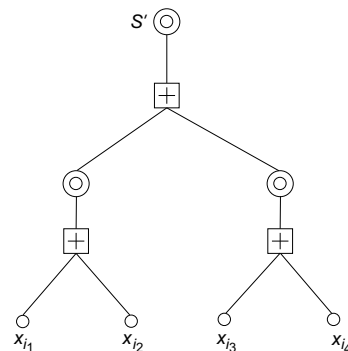


Fig. 3. Balanced subtree of the $z = 4$ least reliable symbol nodes.

pared to the $k - 2$ \boxplus -operations in the full version, the simplified version needs only $z - 2$ \boxplus -operations, to which the small overhead for searching and for the sign operations has to be added.

Note that the partial sorting idea can also be applied to Gallager's decoding approach as described in Section 2. The reliability values for the outgoing messages are then computed from the z -fold \boxplus -sum $L(x_{i_1}) \boxplus L(x_{i_2}) \boxplus \dots \boxplus L(x_{i_z})$.

C. Fast Partial Sorting

The partial-sorting problem, which finds the z smallest elements in a set of k real values, is a problem whose worst-case complexity is difficult to analyze [12]. Here, we give a simple algorithm that provides a simple upper bound on the number of comparisons needed. The algorithm is based on merge-sorting as described above but with the following modification: in the ordered list obtained from the merging of two smaller ordered lists, only the z smallest values are kept and all other (larger) values are deleted from the list. As a result, the maximum list size at each stage of the algorithm is z . A balanced tree with k leaves contains a total of $k - 1$ inner nodes (that are not leaves). At these inner nodes, at most z comparisons have to be done. Hence, the total complexity of partial sorting is upper-bounded by $z(k - 1)$ comparisons.

Note that when the comparisons are done in parallel at each level in the tree, the run time of the algorithm corresponds to about $\log_2(k)$ comparison operations.

V. SIMULATION RESULTS

For simulations on the AWGN channel, we have considered an LDPC code of length $N = 4489$ and rate 4158/4489, which is defined by $M = 335$ parity checks. Fig. 4 shows the bit-error-rate performance of this code. The following LDPC decoding algorithms have been used: the full SPA, the ordered-statistics-based approximations of the SPA with $z = 2, 3$, and 4 least reliable values. The results are obtained using Monte Carlo simulations, in which the maximum number of iterations is fixed to 80 in all cases. For the core operation \boxplus , we use Eq. (3) in its full accuracy.

We observe that the simple min-sum approximation, which essentially corresponds to the ordered-statistics-based approximation of the SPA with only $z = 2$ least reliable values, suffers a performance penalty of about 0.3 dB at a bit-error rate of 10^{-6} . It is apparent from Fig. 4 that the loss in performance is recovered by increasing the number z of remaining least reliable values. For instance, for $z = 3$ the performance loss falls within 0.1 dB of the full SPA, and for $z = 4$ it is only 0.04 dB from the full SPA.

Fig. 5 shows the bit-error-rate performance of another MacKay's LDPC code of length $N = 1008$ and rate 1/2 on an AWGN channel. Again we see that the simple min-sum approximation suffers nonnegligible performance loss relative to the full SPA, which can be regained by using $z = 3$ or $z = 4$.

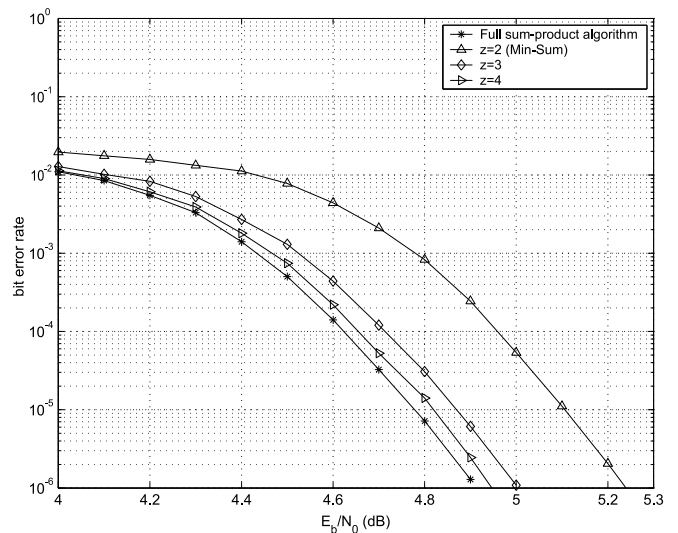


Fig. 4. Performance of the SPA and its simplifications using $z = 2, 3$, and 4 least reliable values for a rate-4158/4489 code on the AWGN channel.

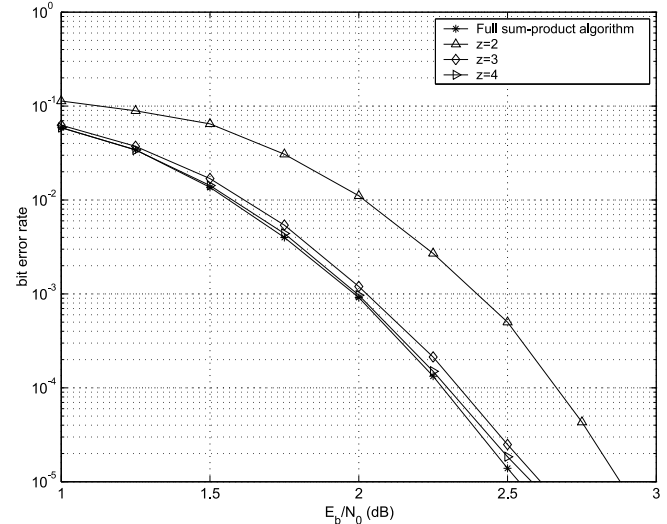


Fig. 5. Performance of the SPA and its simplifications using $z = 2, 3$, and 4 least reliable values for a rate-1/2 code (1008, 504) on the AWGN channel.

VI. CONCLUSIONS

A family of low-complexity and low-latency algorithms that approximate the sum-product algorithm has been proposed. The main feature of the simplification consists in ordering the reliability of the incoming message at each check node. The complexity of the simplified algorithm depends on the number z of least reliable messages that are selected. When keeping only $z = 2$ least reliable messages, the algorithm essentially reduces to the well-known min-sum algorithm, which has the least complexity. Simulation results have shown that for increasing values of z , the performance of the algorithm quickly approaches the performance of the full SPA. By suitably selecting the parameter z , the algorithm provides the flexibility to improve performance at the cost of increased complexity. Moreover, one can simplify the \boxplus -core operation by some reduced-complexity implementation of one's own choice.

ACKNOWLEDGMENTS

We would like to thank Evangelos Eleftheriou and Sedat Ölçer for helpful discussions and comments on an early version of the paper. Moreover, we would like to acknowledge David MacKay of the University of Cambridge for providing the length-4489 and length-1008 LDPC codes.

REFERENCES

- [1] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inform. Theory*, pp. 21–28, 1962.
- [2] S.-Y. Chung, G. D. Forney, Jr., T. Richardson, and R. Urbanke, "On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, Vol. 5, No. 2, pp. 58–60, Feb. 2001.
- [3] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced Complexity Iterative Decoding of Low-Density Parity-Check Codes Based on Belief Propagation," *IEEE Trans. Commun.*, vol. 47, pp. 673–680, May 1999.
- [4] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient Implementations of the Sum-Product Algorithm for Decoding LDPC Codes," in *Proc. Globecom 2001*, San Antonio, TX, USA, pp. 1036A–1036E.
- [5] E. Eleftheriou, T. Mittelholzer, and A. Dholakia, "A Reduced-Complexity Decoding Algorithm for Low-Density Parity-Check Codes," *IEE Electron. Lett.*, Vol. 37, pp. 102–104, Jan. 2001.
- [6] F. R. Kschischang, B. J. Frey, H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. Inform. Theory*, Vol. 47, No. 2, Feb. 2001, pp. 498–519.
- [7] D. J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.
- [8] J. Hagenauer, M. Moerz, and E. Offer, "Analog Turbo-Networks in VLSI: The Next Step in Turbo Decoding and Equalization," in *Proc. 2nd Intl. Symp. Turbo Codes & Related Topics*, Brest, France, Sept. 2000, pp. 209–218.
- [9] H.-A. Loeliger, F. Lustenberger, and M. Helfenstein, and F. Tarkoy, "Probability Propagation and Decoding in Analog VLSI," *IEEE Trans. Inform. Theory*, vol. 47, pp. 837–843, Feb. 2001.
- [10] J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429–445, Mar. 1996.
- [11] S. M. Aji and R. J. McEliece, "The Generalized Distributive Law," *IEEE Trans. Inform. Theory*, Vol. 46, No. 2, pp. 325–343, Mar. 2000.
- [12] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, Vol. 3, Addison-Wesley, 1973.