Stochastic Iterative Decoders

Chris Winstead
ECE Dept.
Utah State University
4120 Old Main Hill
Logan, UT 84322-4120
USA
Fax: (435) 797-3054
Email: *winstead@engineering.usu.edu*

Vincent C. Gaudet, Anthony Rapley, Christian Schlegel
ECE Dept.
2nd Floor ECERF bldg.
University of Alberta
Edmonton, AB T6G 2V4
Canada
Email: *[vgaudet,rapley,schlegel]@ece.ualberta.ca*

***Abstract*— This paper presents a stochastic algorithm for iterative error control decoding. We show that the stochastic decoding algorithm is an approximation of the sum-product algorithm. When the code's factor graph is a tree, as with trellises, the algorithm approaches maximum a-posteriori decoding. We also demonstrate a stochastic approximations to the alternative update rule successive relaxation. Stochastic decoders have very simple digital implementations which have almost no RAM requirements. We present example stochastic decoders for a trellis-based Hamming code, and for a Block Turbo code constructed from Hamming codes.**

I. Introduction

Iterative decoding describes a class of powerful graph-based algorithms for error control decoding, including Turbo and LDPC decoders. Implementations tend to be complex, and much research effort has been invested to produce less complex physical solutions [1]–[4].

We describe implementation strategies in terms of *parallel* and *serial* architectures. The parallel approach requires large numbers of small processors which operate concurrently. This approach consumes a large amount of silicon area.

In serial architectures, processors are reused multiple times within an iteration, thus requiring use of a RAM for storing messages. This approach requires less silicon area, but reduces the decoder's maximum throughput. The serial approach is also expensive to apply in reconfigurable implementations, because RAM tends to be limited in current FPGA platforms.

In this paper, we present a stochastic approximation to the sum-product algorithm. This approximation allows for low-complexity parallel decoder implementations. The resulting circuits have no RAM requirements, and are simple enough to be implemented in low-cost FPGAs.

In stochastic implementations, each processor is implemented by a simple logic gate. The stochastic algorithm can therefore be implemented on a code's factor graph, using only a few gates per node.

Similar algorithms have been presented elsewhere [5]. The stochastic decoding algorithm is related to Pearl's method of "stochastic simulation," which provides accurate belief propagation in cyclic Bayesian networks [6]. Pearl's method does not work for cyclic networks with deterministic constraints, and therefore cannot be used for error-control decoding.

In this paper, we present a new form of the stochastic algorithm which is more general than the original, strictly binary algorithm [5]. Unlike Pearl's method, our goal is not to produce cycle-independent results, but to produce a low-complexity approximation of the sum-product decoding algorithm.

This paper is organized as follows. In Sec. II we present a review of notation and terminology. Sec. II-B summarizes the widely-used *sum-product* iterative update rule. Sec. II-C presents the less common *successive relaxation* update rule, a derivative of sum-product which sometimes yields better performance and/or faster convergence [7].

In Sec. III we describe the new stochastic decoding algorithm. Stochastic implementation on acyclic code graphs is discussed in Sec. III-B, which also presents results for a stochastic trellis decoder.

Cyclic constraint graphs require use of a modified node, called the supernode, which is discussed in Sec. III-C. This Section also presents results for a length-256 stochastic Block Turbo decoder.

Throughout the paper, significant results are presented as theorems. The proofs of these theorems are intended to be illustrative, but not fully rigorous.

II. Message-Passing Decoding Algorithms

The conventional sum-product algorithm consists of *probability propagation* through nodes in a code's *factor graph* [8]. For our purposes, we consider a more limited class of factor graphs called *constraint graphs*.

A constraint graph represents the relationships among a code's *variables* and *constraints*. Constraint functions define which combinations of variables are permissible and which are not.

A. Constraint graphs.

A constraint graph is a type of factor graph which shows the constraints among a code's information and parity bits. The constraint graph consists of nodes and edges. Variables are usually represented by circles, and constraints (functions) are represented by squares.

It is customary to show information and parity bits explicitly as circles in the constraint graph. These variables represent observable information, and have a single edge connection.

A code is constructed from a family of constraint functions, represented in the graph by squares. For most codes of interest, all constraints may be expressed using functions of no more than three variables.

We now proceed to a more precise explanation of these concepts. Let A, B and C be variables with alphabets $\mathcal{A}_A$, $\mathcal{A}_B$ and $\mathcal{A}_C$, respectively. We denote particular values of A, B and C using lower-case letters a, b and c. Without loss of generality, we assume that $\mathcal{A}_A$, $\mathcal{A}_B$ and $\mathcal{A}_C$ are subsets of the natural numbers (including 0).

Let $\mathcal{D}$ be defined by the Cartesian product $\mathcal{D} \doteq \mathcal{A}_A \times \mathcal{A}_B \times \mathcal{A}_C$. Let f be a Boolean function, $f : \mathcal{D} \to \{0, 1\}$, and define the set $S \doteq \{(a,\, b,\, c) \in \mathcal{D} \;:\; f(a,\, b,\, c) = 0\}$. The function f induces mappings f_A, f_B and f_C, defined by

$$f_C(a,\, b) \doteq \{c \in \mathcal{A}_C \;:\; (a,\, b,\, c) \in S\}\,.$$

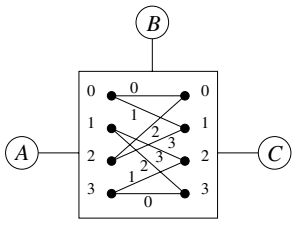
The mappings f_A and f_B are defined similarly.

If f_A, f_B and f_C are all functions, then f is said to be a *constraint function*. For some combination $(a,\, b,\, c)$, if $f(a,\, b,\, c) = 0$ then the constraint is said to be *satisfied*. It is sometimes convenient to refer to the set S as the *satisfaction* of the constraint function f.

A constraint function has a one-to-one correspondence with its satisfaction. The satisfaction can be written as a table with three columns, which we call the *satisfaction table*.

The satisfaction table corresponds to a *trellis graph*, which consists of two columns of states and a set of branches connecting between them. On the left of the trellis, the states correspond to the alphabet $\mathcal{A}_A$. The states on the right correspond to $\mathcal{A}_C$. The branches in the middle are labeled with symbols from $\mathcal{A}_B$. For each row $(a,\, b,\, c)$ in the satisfaction table, there is a branch in the trellis which connects a with c, and which is labeled b. This relationship is illustrated by Fig. 1.

When we present a constraint graph, we include an unlabeled trellis graph for each constraint node. This trellis graph indicates the structure and complexity of the corresponding constraint function.



A.

a	b	c
0	0	0
0	1	1
1	3	2
1	2	3
2	2	0
2	3	1
3	1	2
3	0	3

B.

Fig. 1. An example constraint node (A), showing a detailed trellis description of its constraint; and (B) the set S corresponding to this constraint.

B. The sum-product update rule.

The sum-product algorithm replaces each variable in the constraint graph with a random variable. The task is then to compute the (conditional) probability mass of one variable, given the available independent (conditional) probability masses of all other variables.

Each edge in the graph is then associated with two probability masses, because each edge is connected to two nodes. Each function node produces, for every edge, a unique estimate of the (conditional) probability mass, based on the information which appears at the other edges. Probability masses are generally represented by vectors, and may sometimes be represented by summary messages, such as log-likelihood ratios or soft bits.

Let ρ_A, ρ_B and ρ_C be the conditional probability mass vectors for variables A, B and C, respectively, and let $\rho_A(i)$ denote the i^{th} element of ρ_A. Define $S_{C=c}$ as the set $S_{C=c} \doteq \{(a,\, b) \;:\; (a,\, b,\, c) \in S\}$. For most applications, the sum-product algorithm is described by the equation

$$\rho_C(c) \propto \sum_{(a,b)\in S_{C=c}} \rho_A(a)\,\rho_B(b)\,. \tag{1}$$

The proportionality symbol in (1) is used to indicate that the resulting mass vector must be normalized so that the sum is equal to one. Let S_C be the union $S_C \doteq \bigcup_{c\in\mathcal{A}_C} S_{C=c}$. The normalized sum-product equation is then

$$\begin{aligned}\rho_C(c) &= \frac{\sum_{(a,b)\in S_{C=c}} \rho_A(a)\,\rho_B(b)}{\sum_{(a,b)\in S_C} \rho_A(a)\,\rho_B(b)} \\ &= \frac{\sum_{(a,b)\in S_{C=c}} \rho_A(a)\,\rho_B(b)}{1 - \sum_{(a,b)\in \overline{S_C}} \rho_A(a)\,\rho_B(b)}.\end{aligned} \tag{2}$$

The node update rule (2) is used to update the message transmitted on each edge from each function node.

Throughout this paper, we will assume the use of a "flooding" schedule for message passing. In this schedule, all nodes receive messages at the start of each iteration. They compute updated messages on each edge, and then complete the iteration by transmitting the updated messages.

C. The relaxation update rule.

In the conventional sum-product algorithm, a node's outgoing messages are *replaced* by a new probability mass determined according to (2). One alternative to replacement is the method of successive relaxation [7], which we refer to simply as *relaxation*.

Let v_C be the mass estimated according to (2). Also let ρ_C refer to the mass produced by relaxation at time t, and let ρ'_C be the mass produced by relaxation at time $t-1$. Then the relaxation update rule is described by

$$\rho_C = \rho'_C + \beta\,(\nu_C - \rho'_C)\,. \tag{3}$$

where β is referred to as the *relaxation parameter*, $0 < \beta < 1$.

The relaxation method produces smoother updating than the replacement method. It has been shown that relaxation (3) is equivalent to Euler's method for simulating differential

equations. It has also been shown that the relaxation method results in superior performance for some codes.

III. The Stochastic Algorithm

The stochastic decoding algorithm is a message-passing algorithm. Like the sum-product algorithm, it is based on the code's constraint graph. In the stochastic algorithm, messages are not probability mass vectors. Stochastic messages are integers which change randomly according to some probability mass.

Stochastic messages can be regarded as sequences of integers. The probability mass is communicated over time by the sequence. One mass corresponds to many possible sequences.

At each discrete time instant t, a function node, N, receives messages $A_r(t)$, $B_r(t)$ and $C_r(t)$, and transmits messages $A_t(t)$, $B_t(t)$ and $C_t(t)$. The integer messages vary randomly over time, according to the probability masses ρ_A, ρ_B and ρ_C, respectively. The message $A_r(t)$ therefore equals integer a with probability $\rho_A(a)$. All probabilities lie in the open interval $(0, 1)$.

A. The stochastic update rule.

To implement the stochastic message-passing algorithm, we use the following deterministic message update rule at each function node. We consider the propagation of message from $(A_r(t), B_r(t))$ to $C_t(t)$. Suppose that, at time t, $A_r(t) = a$ and $B_r(t) = b$. Then

$$C_t(t) = \begin{cases} f_C(a, b) & \text{if } (a, b) \in S \\ C_t(t-1) & \text{otherwise.} \end{cases} \quad (4)$$

We claim that the stochastic update rule (4) produces a message C with mass ρ_C which is equal to the mass computed by the sum-product update rule (2). To support this claim, we rely upon the following assumptions:

1) There is no correlation between any pair of sequences received by a constraint node.
2) The probability mass associated with any message sequence is *stationary*.

When a node is considered in isolation from the graph, the above conditions apply.

Theorem 1: A node N receives stochastic messages $A_r(t)$ and $B_r(t)$, and transmits a message $C_t(t)$ according to the update rule (4). The received messages have stationary probability masses ρ_A and ρ_B. Then the mass of the transmitted message, ρ_C, is equal to the result computed by the sum-product update rule.

Proof: The probability of transmitting a particular integer c is

$$\rho_C(c) = \sum_{(a,b)\in S_{C=c}} \rho_A(a)\rho_B(b) + \left[\sum_{(a,b)\in\overline{S_C}} \rho_A(a)\rho_B(b)\right] \Pr(C_t(t-1) = c).$$

We assume that the mass of each variable is stationary. Then $\Pr(C_t(t-1) = c) = \rho_C(c)$. Therefore

$$\rho_C(c) = \frac{\sum_{(a,b)\in S_{C=c}} \rho_A(a)\rho_B(b)}{1 - \sum_{(a,b)\in\overline{S_C}} \rho_A(a)\rho_B(b)},$$

which is the same as (2). ∎

The stochastic update rule (4) results in very simple node implementations. The implementation of a node consists of a logic gate which is precisely as complex as the node's trellis description.

B. Message-passing implementation on acyclic graphs.

For a constraint graph with no cycles, a complete stochastic decoder is constructed by applying the update rule (4) to every constraint node. Probability masses, representing the channel information, are stored at the variable nodes. The variable nodes generate the first messages randomly, according to the channel information.

The random integer messages flow deterministically through the decoder, through a cascade of constraint nodes, until they arrive back at the variable nodes. The messages are tabulated in histograms as they arrive at the variable nodes. After l time-steps, the variable nodes make decisions by selecting the symbol with the largest count.

When implemented in this way, the assumptions of Sec. III-A are never violated. The probability mass of each message in the decoder is therefore equivalent to that of the sum-product algorithm. By increasing l, the result of stochastic decoding can be made arbitrarily close to that of sum-product decoding.

Example 1: An acyclic constraint graph for a (16, 11) Hamming code is shown in Fig. 2. A stochastic decoder is constructed by replacing each constraint node with logic gates, according to the update rule (4). We allow $l = 250$ time-steps for decoding. The performance of the resulting code for antipodal transmission on a Gaussian channel is shown in Fig. 3.

C. Graphs with cycles and Supernodes.

If the constraint graph contains one or more cycles, then the assumptions of Sec. III-A are violated. In particular, the independence among received messages $A_r(t)$, $B_r(t)$ and $C_r(t)$ no longer holds for some or all constraint nodes.

When the messages at a node are correlated, it is possible for a group of nodes to settle into a fixed state. This state is maintained solely by the messages within the cycle. No pattern of independent messages is sufficient to restore the cycle to proper functioning. We refer to this phenomenon as *latching*.

A simple latching example is the all-zero state in an LDPC graph. If all internal messages between parity and equality nodes are zero, then they will be held permanently at zero, regardless of any activity from the variable nodes.

To avoid latching, cycles may be interrupted with a special stochastic node, called a *supernode*. A supernode receives stochastic messages and tabulates them in histograms. It then generates new messages based on the estimated probability masses, resulting in a new uncorrelated sequence.

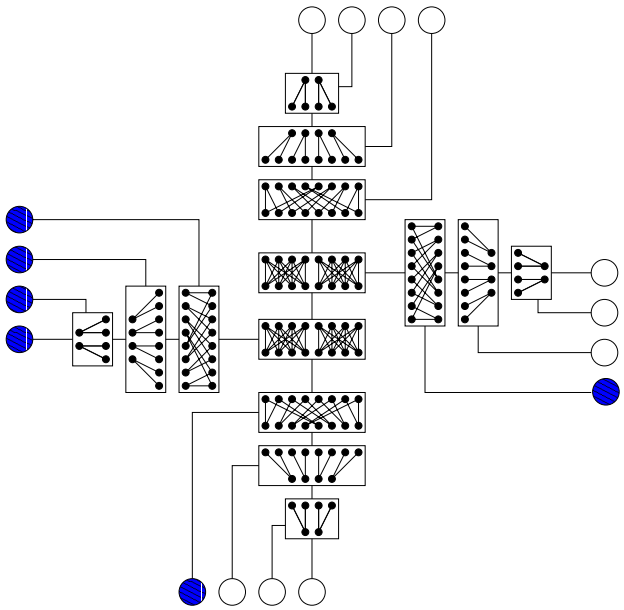


Fig. 2. An acyclic constraint graph for the (16, 11) Hamming code. Dark circles represent parity bits. Light circles represent information bits.

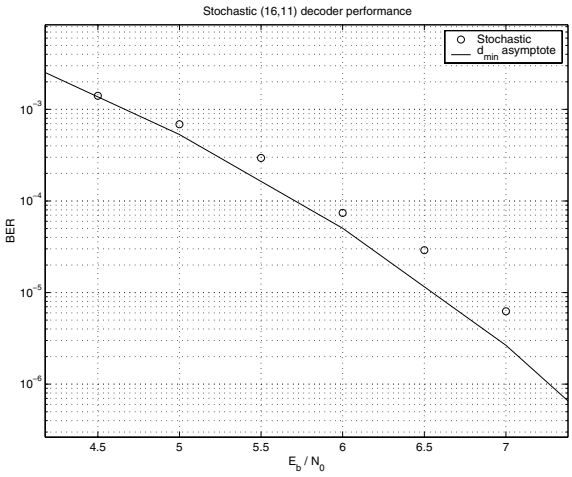


Fig. 3. Performance results for a (16,11) Hamming stochastic decoder, based on the graph in Fig. 2. Also shown is the code's minimum-distance asymptote. Each data point represents 50 errors.

A supernode can be implemented with counters and a linear feedback shift-register (LFSR) to generate random numbers. The simplest supernode implementation is illustrated in Fig. 4.

We also claim that the supernode must be *packetized*, which is defined as follows. Suppose a supernode is generating a random sequence according to an estimated mass ρ. We say that the supernode's behavior is packetized if the estimated mass ρ is only updated every l time-steps. During these l time-steps, the supernode tabulates the received message(s). After the l^{th} time-step, ρ is updated with the newly tabulated information.

Data received during the l time-steps is called a *packet*, and the l-step time interval is called an *iteration*. There are two

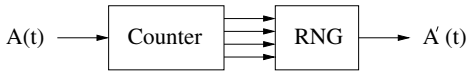


Fig. 4. A simple supernode implementation. A stochastic sequence $A(t)$ is converted into an uncorrelated sequence $A'(t)$ which encodes the same mass. 'RNG' denotes a random number generator, which generates random numbers according to the mass estimated for $A(t)$.

obvious procedures by which ρ may be updated:

1) The tabulation is cleared after each iteration, and the mass estimate is *replaced* by a new estimate representing only the data from the most recent packet.
2) The tabulation is not cleared after each packet of l time-steps. The tabulation is *accumulated* over many iterations until decoding is complete.

For sufficiently large l, these supernode update rules approach the behavior of sum-product and relaxation updates, respectively.

Theorem 2: Suppose each cycle in a stochastic constraint graph contains a supernode, and the mass estimated by the supernode is updated by *replacement* every l time-steps. Then the result of stochastic decoding approaches that of sum-product decoding as l is increased.

Proof: Let l be the total number of observations of an event C, and let $T_C(c)$ be the total number of observations for which $C = c$. By definition for a stationary mass ρ, $\lim_{l\to\infty} T_C(c)/l = \rho(c)$. The mass of C is is therefore estimated with arbitrarily small error as l is increased.

The messages produced by the supernode have a fixed mass, and are therefore stationary. They are also not correlated with any other messages in the graph. Theorem 1 therefore applies, and the stochastic decoder approaches the behavior of a sum-product iteration for large l. ■

Theorem 3: Suppose each cycle in a stochastic constraint graph contains a supernode, and the mass estimated by the supernode is updated by *accumulation* every l time-steps. Then the result of stochastic decoding approaches that of relaxation, where the relaxation parameter β varies as $1/m$, where m is the number of iterations.

Proof: Let ρ_C be a supernode's newly updated estimate of the mass of an event C just after m iterations, and let ρ'_C be the supernode's estimate after $m-1$ iterations. The supernode has made ml total observations. Let N_c be number of observations in the m^{th} packet for which $C = c$. Let N'_c be the number of observations in the first $m-1$ packets for which $C = c$. Then the new estimate is

$$\begin{aligned}\rho_C &= \frac{N_c + N'_c}{ml} \\ &= \frac{1}{m}\left(\frac{N_c}{l}\right) + \frac{m-1}{m}\left(\frac{N'_c}{(m-1)\,l}\right) \\ &= \rho'_C + \frac{1}{m}\left(\frac{N_c}{l} - \rho'_C\right). \end{aligned} \tag{5}$$

As $l \to \infty$, N_c/l approaches the sum-product estimate of ρ_c for the m^{th} iteration. Upon substitution, we find that (5)

approaches the relaxation update rule with $\beta = 1/m$. The difference between relaxation and (5) can be made arbitrarily small by increasing l. ■

Example 2: The (16,11) Hamming code graph of Ex. 1 can be used to construct a (256,121) Block Turbo Code. The decoder for this code consists of 32 Hamming decoders, arranged in 16 rows and 16 columns. Each row shares precisely one bit with each column. Where two decoders share a bit, they are joined by an *equality* constraint.

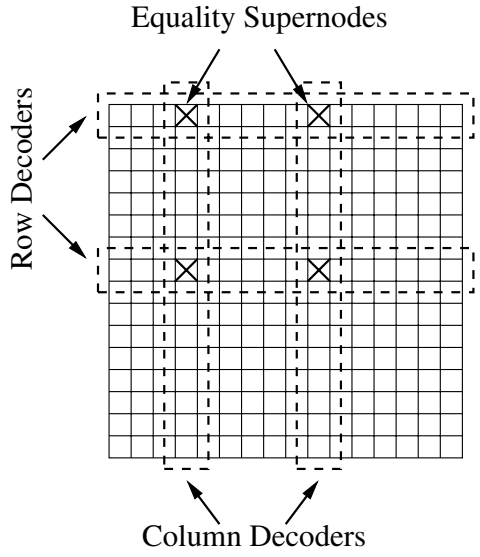


Fig. 5. The structure of a product code. Each row represents an instance of the decoder from Fig. 2. Each column represents another instance of the decoder. Where a row and a column cross, they are joined by an equality supernode. The code's variable nodes also connect to the equality supernode.

The equality constraints create cycles in the constructed graph. To resolve this, we implement each equality node as a supernode. Within the equality supernode, received messages are converted to probability mass estimates. Using these mass estimates, the outgoing masses are computed using the conventional sum-product update rule (2). New stochastic sequences are generated to conform to the outgoing mass estimate.

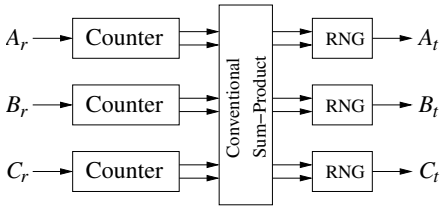


Fig. 6. Structure of the equality supernode. The counters update by accumulation. The conventional sum-product calculation is invoked every 250 time-steps.

We choose a flooding schedule with an *accumulation* update rule, and a packet length of $l = 250$. Eight iterations are allowed, for a total decoding time of 2000 time-steps. The stochastic decoder's performance is as shown in Fig. 7. Also shown is the simulated performance of a commercial Block Turbo decoder from Comtech/Advanced Hardware Architectures [9]. The AHA decoder uses 26 iterations for decoding, and six bits of precision for the channel samples.

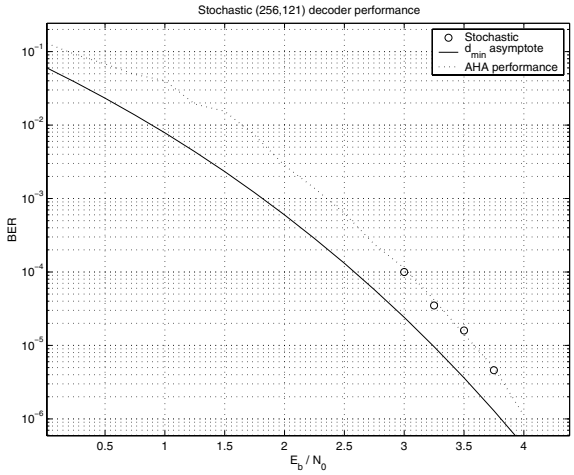


Fig. 7. Performance of a stochastic (256,121) Block Turbo decoder. Also shown is simulated data for a similar Block Turbo decoder produced by Comtech AHA. Each data point represents 50 errors.

IV. Discussion and Conclusions

While this paper validates that stochastic iterative decoding can be applied to interesting codes (e.g., Block Turbo codes), it also raises an interesting set of ancillary questions. There are numerous alternative stochastic update rules and message schedules, and many possible simplifications to the algorithm. Questions also remain concerning the relationships among the packet size, the number of iterations and the performance of a stochastic decoder for a given graph.

These issues hint at a broad set of interesting research questions, with significant potential to improve the complexity of iterative decoders.

References

[1] M. Bekooij, J. Dielissen, F. Harmsze, S. Sawitzki, J. Huisken, A. van der Werf, and J. van Meerbergen, "power-efficient application-specific VLIW processor for Turbo decoding," in *Proc. 2001 IEEE International Solid State Circuits Conference (ISSCC'01)*, Feb. 2001, pp. 180–181.

[2] J. Hagenauer and M. Winklhofer, "The analog decoder," *Proc. International Symposium on Information Theory*, Aug. 1998.

[3] Jagadeesh Kaza and Chaitali Chakrabarti, "Design and implementation of low-energy Turbo decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 9, pp. 968–977, Sept. 2004.

[4] H. A. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarkoy, "Probability propagation and decoding in analog VLSI," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 837–843, Feb. 2001.

[5] Vincent C. Gaudet and Anthony C. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, vol. 39, no. 3, pp. 299–301, feb 2003.

[6] Judea Pearl, *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*, Morgan Kaufmann, 1988.

[7] Saied Hemati and Amir H. Banihashemi, "On the dynamics of continuous-time analog iterative decoding," in *Proc. 2004 Int'l Symposium on Information Theory (ISIT 04)*, Chicago, June 2004, p. 262.

[8] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[9] Comtech AHA, "http://www.aha.com," 2004.