An Introduction to Factor Graphs

Hans-Andrea Loeliger
ETH Zürich

Abstract—A large variety of algorithms in coding, signal processing, and artificial intelligence may be viewed as instances of the *summary-product algorithm* (or *belief/probability propagation algorithm*), which operates by *message passing* in a graphical model. Specific instances of such algorithms include Kalman filtering and smoothing, the forward-backward algorithm for hidden Markov models, probability propagation in Bayesian networks, and decoding algorithms for error correcting codes such as the Viterbi algorithm, the BCJR algorithm, and the iterative decoding of turbo codes, low-density parity check codes, and similar codes. New algorithms for complex detection and estimation problems can also be derived as instances of the summary-product algorithm. In this paper, we give an introduction to this unified perspective in terms of (Forney-style) factor graphs.

1 Introduction

Engineers have always liked graphical models such as circuit diagrams, signal flow graphs, trellis diagrams, and a variety of block diagrams. In artificial intelligence, statistics, and neural networks, stochastic models are often formulated as Bayesian networks or Markov random fields. In coding theory, the iterative decoding of turbo codes and similar codes may also be understood in terms of a graphical model of the code.

Graphical models are often associated with particular algorithms. For example, the Viterbi decoding algorithm is naturally described by means of a trellis diagram, and estimation problems in Markov random fields are often solved by Gibbs sampling.

This paper is an introduction to *factor graphs* and to the associated *summary propagation algorithms*, which operate by passing "messages" ("summaries") along the edges of the graph. The origins of factor graphs lie in coding theory, but they offer an attractive notation for a wide variety of signal processing problems. In particular, a large number of practical algorithms for a wide variety of detection and estimation problems can be derived as summary propagation algorithms. The algorithms derived in this way often include the best previously known algorithms as special cases or as obvious approximations.

The two main summary propagation algorithms are the *sum-product* (or *belief propagation* or *probability propagation*) algorithm and the *max-product* (or *min-sum*) algorithm, both of which have a long history. In the context of error correcting codes, the sum-product algorithm was invented by Gallager [17] as a decoding algorithm for low-density

H.-A. Loeliger is with the Dept. of Information Technology and Electrical Engineering, ISI-ITET, ETH Zürich, CH-8092 Zürich, Switzerland. Email: loeliger@isi.ee.ethz.ch.

parity check (LDPC) codes; it is still the standard decoding algorithm for such codes. However, the full potential of LDPC codes was not yet realized at that time. Tanner [41] explicitly introduced graphs to describe LDPC codes, generalized them (by replacing the parity checks with more general component codes), and introduced the min-sum algorithm.

Both the sum-product and the max-product algorithm have also another root in coding, viz. the BCJR algorithm [5] and the Viterbi algorithm [10], which both operate on a trellis. Before the invention of turbo coding, the Viterbi algorithm used to be the workhorse of many practical coding schemes. The BCJR algorithm, despite its equally fundamental character, was not widely used; it therefore lingered in obscurity and was independently re-invented several times.

The full power of iterative decoding was only realized by the breakthrough invention of turbo coding by Berrou et al. [6], which was followed by the rediscovery of LDPC codes [33]. Wiberg et al. [45], [46] observed that the decoding of turbo codes and LDPC codes as well as the Viterbi and BCJR algorithms are instances of one single algorithm, which operates by message passing in a generalized Tanner graph. From this perspective, new applications such as, e.g., iterative decoding for channels with memory also became obvious. The later introduction of factor graphs [15], [24] may be viewed as a further elaboration of the ideas by Wiberg et al. In the present paper, we will use *Forney-style* factor graphs, which were introduced in [13] (and there called "normal graphs").

Meanwhile, the work of Pearl and others [38], [49], [50], [26] on probability propagation (or belief propagation) in Bayesian networks had attracted much attention in artificial intelligence and statistics. It was therefore exciting when, in the wake of turbo coding, probability propagation and the sum-product algorithm were found to be the same thing [14], [4]. In particular, the example of iterative decoding proved that probability propagation, which had been used only for cycle-free graphs, could be used also for graphs with cycles.

In signal processing, both hidden-Markov models (with the associated forward-backward algorithm) and Kalman filtering (especially in the form of the RLS algorithm) have long been serving as workhorses in a variety of applications, and it had gradually become apparent that these two techniques are really the same abstract idea in two specific embodiments. Today, these important algorithms may be seen as just two other instances of the sum-product (probability propagation) algorithm. In fact, it was shown in [24] (see also [4]) that even fast Fourier transform (FFT) algorithms may be viewed as instances of the sum-product algorithm.

Graphical models such as factor graphs support a general trend in signal processing from *sequential* processing to *iterative* processing. In communications, for example, the advent of turbo coding has completely changed the design of receivers; formerly sequentially arranged subtasks such as synchronization, equalization, and decoding are now designed to interact via multiple feedback loops. Another example of this trend are "factorial hidden Markov models" [18], where the state space of traditional hidden Markov models is split into the product of several state spaces. Again, virtually all such signal processing schemes are examples of summary propagation and may be systematically derived from suitable factor graphs.

The literature on graphical models and their applications is vast. The references mentioned in this paper are a somewhat arbitrary sample, very much biased by the author's personal perspective and interests. Some excellent papers on iterative coding and communications are contained in the special issues [1], [2], [3]; beautiful introductions to codes on graphs and the corresponding algorithms are also given in [11], [12], [25]. Much of the literature on graphical models appears under the umbrella of neural networks, cf. [22]. A much expected survey on graphical models other than factor graphs is the book by Jordan [23].

This paper is structured as follows. In Section 2, we introduce factor graphs. (In the main ideas, we will follow [24] and [13], but we will also adopt some details of notation from [27] and [42].) The use of such graphs for error correcting codes is described in Section 3. In Section 4.1, the pivotal issue of eliminating internal variables from a model is considered. The summary-product algorithm is introduced in Section 4.2. The wide area of signal processing by message passing is briefly addressed in Sections 4.3 and 4.4. Some further topics, ranging from convergence issues to analog realizations of the sum-product algorithm, are briefly touched in Section 5, and some conclusions are offered in Section 6.

2 Factor Graphs

As mentioned, we will use *Forney-style* factor graphs (FFGs) rather than the original factor graphs of [24] (cf. the box on page 9). An FFG is a diagram as in Fig. 1 that represents the factorization of a function of several variables. Assume, for example, that some function $f(u, w, x, y, z)$ can be factored as

$$f(u, w, x, y, z) = f_1(u, w, x) f_2(x, y, z) f_3(z). \tag{1}$$

This factorization is expressed by the FFG shown in Fig. 1. In general, an FFG consists of nodes, edges, and "half edges" (which are connected only to one node), and the FFG is defined by the following rules:

- There is a (unique) node for every factor.
- There is a (unique) edge or half edge for every variable.
- The node representing some factor g is connected with the edge (or half edge) representing some variable x if and only if g is a function of x.

Implicit in this definition is the assumption that no variable appears in more than two factors. We will see below how this seemingly severe restriction is easily circumvented.

The factors are sometimes called *local functions* and their product is called the *global function.* In (1), the global function is f, and f_1, f_2, f_3 are the local functions.

A *configuration* is a particular assignment of values to all variables. The *configuration space* Ω is the set of all configurations; it is the domain of the global function f. For

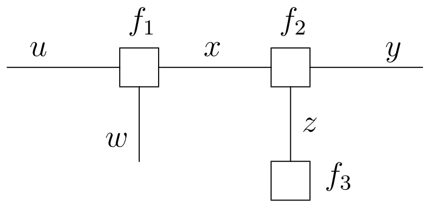


Figure 1: A (Forney-style) factor graph (FFG).

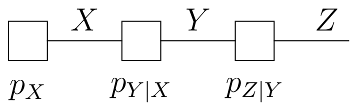


Figure 2: FFG of a Markov chain.

example, if all variables in Fig. 1 are binary, the configuration space Ω is the set $\{0,1\}^5$ of all binary 5-tuples; if all variables in Fig. 1 are real, the configuration space is $\mathbb{R}^5$.

We will primarily consider the case where f is a function from Ω to $\mathbb{R}^+$, the set of nonnegative real numbers. In this case, a configuration $\omega \in \Omega$ will be called *valid* if $f(\omega) \neq 0$.

In every fixed configuration $\omega \in \Omega$, every variable has some definite value. We may therefore consider also the variables in a factor graph as functions with domain Ω. Mimicking the standard notation for random variables, we will denote such functions by capital letters. E.g., if x takes values in some set $\mathcal{X}$, we will write

$$X : \Omega \to \mathcal{X} : \omega \mapsto x = X(\omega). \tag{2}$$

A main application of factor graphs are probabilistic models. (In this case, the sample space can usually be identified with the configuration space Ω.) For example, let X, Y, and Z be random variables that form a Markov chain. Then their joint probability density (or their joint probability mass function) $p_{XYZ}(x,y,z)$ can be written as

$$p_{XYZ}(x,y,z) = p_X(x)\, p_{Y|X}(y|x)\, p_{Z|Y}(z|y). \tag{3}$$

This factorization is expressed by the FFG of Fig. 2.

If the edge Y is removed from Fig. 2, the remaining graph consists of two unconnected components, which corresponds to the Markov property

$$p(x,z|y) = p(x|y)p(z|y). \tag{4}$$

In general, it is easy to prove the following:

Cut-Set Independence Theorem: Assume that an FFG represents the joint probability distribution (or the joint probability density) of several random variables.

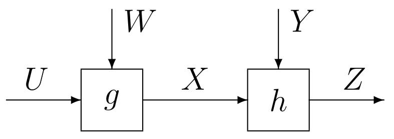


Figure 3: A block diagram.

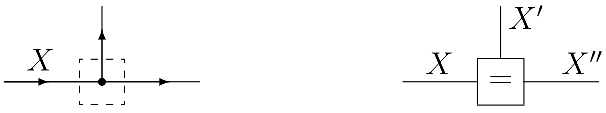


Figure 4: Branching point (left) becomes an equality constraint node (right).

> Assume further that the edges corresponding to some variables $Y_1, \ldots, Y_n$ form a cut-set of the graph (i.e., removing these edges cuts the graph into two unconnected components). In this case, *conditioned* on $Y_1 = y_1, \ldots, Y_n = y_n$ (for any fixed $y_1, \ldots, y_n$), every random variable (or every set of random variables) in one component of the graph is independent of every random variable (or every set of random variables) in the other component.

This fact may be viewed as the "easy" direction of the Hammersley-Clifford Theorem for Markov random fields [47, Ch. 3].

A deterministic block diagram may also be viewed as a factor graph. Consider, for example, the block diagram of Fig. 3, which expresses the two equations

$$X = g(U, W) \tag{5}$$

$$Z = h(X, Y). \tag{6}$$

In the factor graph interpretation, the function block $X = g(U, W)$ in the block diagram is interpreted as representing the factor $\delta\big(x - g(u, w)\big)$, where $\delta(.)$ is the Kronecker delta function if X is a discrete variable or the Dirac delta if X is a continuous variable. (The distinction between these two cases is usually obvious in concrete examples.) Considered as a factor graph, Fig. 3 thus expresses the factorization

$$f(u, w, x, y, z) = \delta\big(x - g(u, w)\big) \cdot \delta\big(z - h(x, y)\big). \tag{7}$$

Note that this function is nonzero (i.e., the configuration is valid) if and only if the configuration is consistent with both (5) and (6).

As in this example, it is often convenient to draw a factor graph with arrows on the edges (cf. Figures 6 and 7).

A block diagram usually contains also branching points as shown in Fig. 4 (left). In the corresponding FFG, such branching points become factor nodes on their own, as is

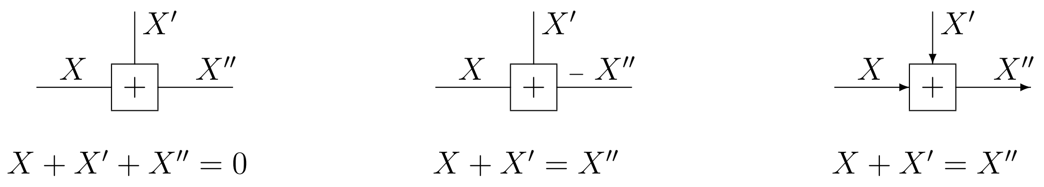


Figure 5: Zero-sum constraint node.

illustrated in Fig. 4 (right). In doing so, there arise new variables (X' und X'' in Fig. 4) and a new factor

$$f_{=}(x, x', x'') \triangleq \delta(x - x')\delta(x - x'') \tag{8}$$

where, as above, $\delta(.)$ denotes either a Kronecker delta or a Dirac delta, depending on the context. Note that $X = X' = X''$ holds for every valid configuration. By this device of variable "cloning", it is always possible to enforce the condition that a variable appears in at most two factors (local functions).

Special symbols are also used for other frequently occuring local functions. For example, we will use the zero-sum constraint node shown left in Fig. 5, which represents the local function

$$f_{+}(x, x', x'') \triangleq \delta(x + x' + x''). \tag{9}$$

Clearly, $X + X' + X'' = 0$ holds for every valid configuration. Both the equality constraint and the zero-sum constraint can obviously be extended to more than three variables.

The constraint $X + X' = X''$ or, equivalently, the factor $\delta(x + x' - x'')$ may be expressed, e.g., as in Fig. 5 (middle) by adding a minus sign to the X'' port. In a block diagram with arrows on the edges, the node in Fig. 5 (right) also represents the constraint $X + X' = X''$.

The FFG in Figure 6 with details in Fig. 7 represents a standard discrete-time linear state space model

$$X[k] = AX[k-1] + BU[k] \tag{10}$$
$$Y[k] = CX[k] + W[k], \tag{11}$$

with $k \in \mathbb{Z}$, where $U[k]$, $W[k]$, $X[k]$, and $Y[k]$ are real vectors and where A, B, and C are matrices of appropriate dimensions. If both $U[.]$ and $W[.]$ are assumed to be white Gaussian ("noise") processes, the corresponding nodes in these figures represent Gaussian probability distributions. (For example, if $U[k]$ is a scalar, the top left node in Fig. 6 represents the function $\frac{1}{\sqrt{2\pi}\sigma}\exp(\frac{-u[k]^2}{2\sigma^2})$.) The factor graph of Fig. 6 and 7 then represents the joint probability density of all involved variables.

In this example, as in many similar examples, it is easy to pass from *a priori* probabilities to *a posteriori* probabilities: if the variables $Y[k]$ are observed, say $Y[k] = y[k]$, then these variables become constants; they may be absorbed into the involved factors and the corresponding branches may be removed from the graph.

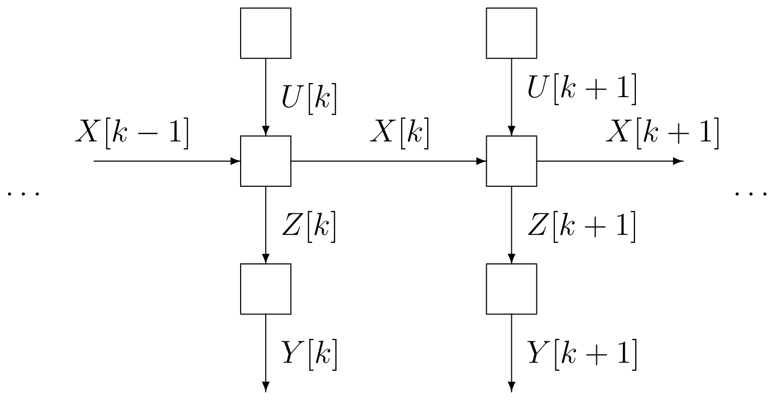


Figure 6: Classical state space model.

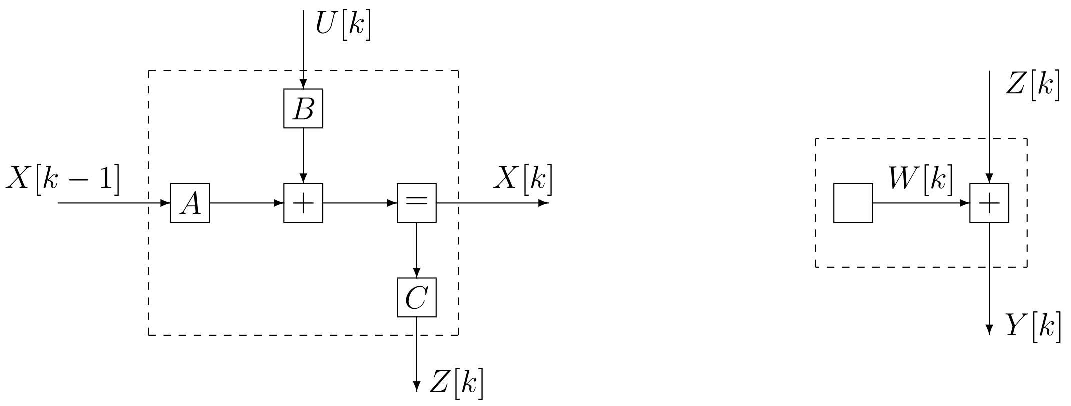


Figure 7: Details of classical linear state space model.

In most applications, we are interested in the global function only up to a scale factor. (This applies, in particular, if the global function is a probability mass function.) We may then play freely with scale factors in the local functions. Indeed, the local functions are often *defined* only up to a scale factor. In this case, we would read Fig. 1 as expressing

$$f(u,w,x,y,z) \propto f_1(u,w,x)f_2(x,y,z)f_3(z) \tag{12}$$

instead of (1), where "$\propto$" denotes equality up to a scale factor.

As exemplified by Figures 6 and 7, FFGs naturally support hierarchical modeling ("boxes within boxes"). In this context, the distinction between "visible" external variables and "hidden" internal variables (state variables) is often important. In an FFG, external variables are represented by half edges, and full edges represent state variables. If some "big" system is represented as an interconnection of subsystems, the connecting edges/variables are internal to the big system but external to (i.e., half edges of) the involved subsystems.

The operation of "closing the box" around some subsystem, i.e., the elimination of internal variables, is of central importance both conceptually and algorithmically. We will return to this issue in Section 4.

3 Graphs of Codes

An error correcting *block code* of length n over some alphabet A is a subset C of A^n, the set of n-tuples over A. A code is *linear* if $A = F$ is a field (usually a *finite* field) and C is a sub*space* of the vector space F^n. A *binary* code is a code with $F = F_2$, the set $\{0,1\}$ with modulo-2 arithmetic. By some venerable tradition in coding theory, the elements of F^n are written as *row* vectors. By elementary linear algebra, any linear code can be represented both as

$$C = \{x \in F^n : Hx^T = 0\} \tag{13}$$

and as

$$C = \{uG : u \in F^k\}, \tag{14}$$

where H and G are matrices over F and where k is the dimension of C (as a vector space over F). A matrix H as in (13) is called a *parity check matrix* for C, and a $k \times n$ matrix G as in (14) is called a *generator matrix* for C. Equation (14) may be interpreted as an *encoding rule* that maps a vector $u \in F^k$ of *information symbols* into the corresponding codeword $x = uG$.

Consider, for example, the binary $(7,4,3)$ Hamming code. (The notation "$(7,4,3)$" means that the code has length $n = 7$, dimension $k = 4$, and minimum Hamming distance 3.) This code may be defined by the parity check matrix

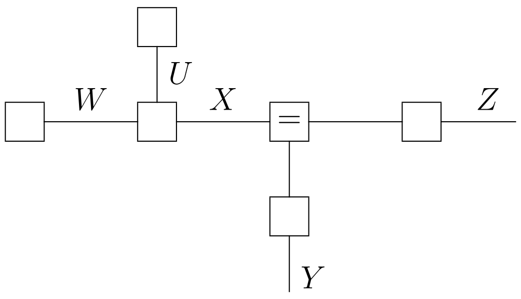
$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}. \tag{15}$$

Other Graphical Models

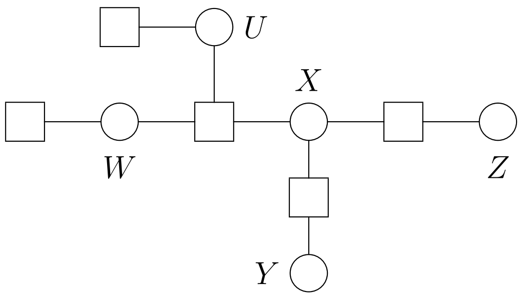
The figures below show the representation of the factorization

$$p(u, w, x, y, z) = p(u)p(w)p(x|u, w)p(y|x)p(z|x)$$

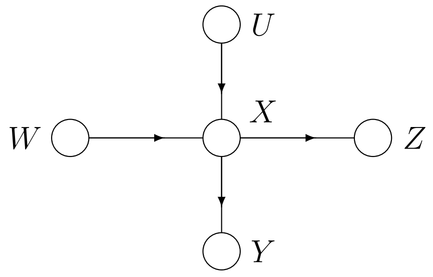
in four different graphical models.



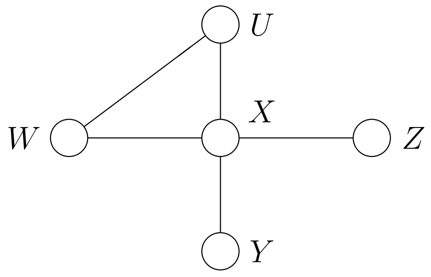
Forney-style factor graph (FFG).



Factor graph as in [24].



Bayesian network.



Markov random field (MRF).

Advantages of FFGs:

- Suited for hierarchical modeling ("boxes within boxes").
- Compatible with standard block diagrams.
- Simplest formulation of the summary-product message update rule.
- Natural setting for Forney's results on Fourier transforms and duality.

It follows from (13) and (15) that the membership indicator function

$$I_C : F^n \to \{0,1\} : x \mapsto \begin{cases} 1, & \text{if } x \in C \\ 0, & \text{else} \end{cases} \tag{16}$$

of this code may be written as

$$I_C(x_1, \ldots, x_n) = \delta(x_1 \oplus x_2 \oplus x_3 \oplus x_5) \cdot \delta(x_2 \oplus x_3 \oplus x_4 \oplus x_6) \cdot \delta(x_3 \oplus x_4 \oplus x_5 \oplus x_7) \tag{17}$$

where $\oplus$ denotes addition modulo 2. Note that each factor in (17) corresponds to one row of the parity check matrix (15).

By a factor graph for some code C, we mean a factor graph for (some factorization of) the membership indicator function of C. (Such a factor graph is essentially the same as a *Tanner graph* for the code [41], [45].) For example, from (17), we obtain the FFG shown in Fig. 8.

The above recipe to construct a factor graph (Tanner graph) from a parity check matrix works for any linear code. However, not all factor graphs for a linear code can be obtained in this way.

The *dual code* of a linear code C is $C^\perp \triangleq \{y \in F^n : y \cdot x^T = 0 \text{ for all } x \in C\}$. The following theorem (due to Kschischang) is a special case of a sweepingly general result on the Fourier transform of an FFG [13] (cf. Section 5).

Duality Theorem for Binary Linear Codes: Consider an FFG for some binary linear code C. Assume that the FFG contains only parity check nodes and equality constraint nodes, and assume that all code symbols $x_1, \ldots, x_n$ are *external* variables (i.e., represented by half edges). Then an FFG for the dual code $C^\perp$ is obtained from the original FFG by replacing all parity check nodes with equality constraint nodes and vice versa.

For example, Fig. 9 shows an FFG for the dual code of the $(7,4,3)$ Hamming code.

A factor graph for a code may also be obtained as an abstraction and generalization of a trellis diagram. For example, Fig. 10 shows a trellis for the $(7,4,3)$ Hamming code. Each codeword corresponds to a path from the leftmost node to the rightmost node, where only movements towards the right are permitted; the codeword is read off from the branch labels along the path. In the example of Fig. 10, a codeword thus read off from the trellis is ordered as $(x_1, x_4, x_3, x_2, x_5, x_6, x_7)$; both this permutation of the variables and the "bundling" of X_2 with X_5 in Fig. 10 lead to a simpler trellis.

Also shown in Fig. 10 is an FFG that may be viewed as an abstraction of the trellis diagram. The variables $S_1, \ldots, S_6$ in the FFG correspond to the trellis states. The nodes in the FFG are $\{0,1\}$-valued functions that indicate the allowed triples of left state, code symbols, and right state. For example, if the trellis states at depth 1 (i.e., the set $\{S_1(\omega) : \omega \in \Omega\}$, the range of S_1) are labeled 0 and 1 (from bottom to top), and if the trellis states at depth 2 (the range of S_2) are labeled 0,1,2,3 (from bottom to top), then the factor $f(s_1, x_4, s_2)$ in the FFG is

$$f(s_1, x_4, s_2) = \begin{cases} 1, & \text{if } (s_1, x_4, s_2) \in \{(0,0,0), (0,1,2), (1,1,1), (1,0,3)\} \\ 0, & \text{else.} \end{cases} \tag{18}$$

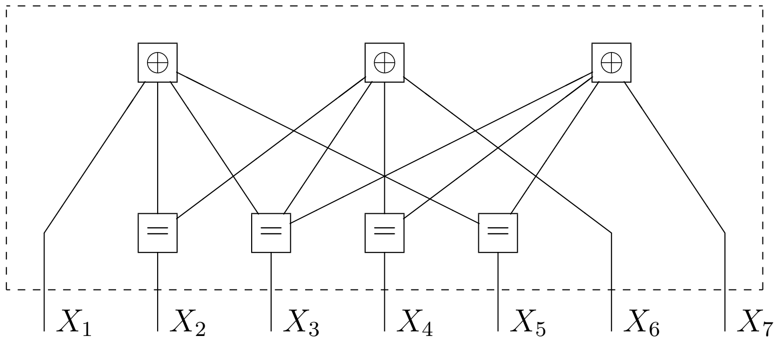


Figure 8: An FFG for the $(7,4,3)$ binary Hamming code.

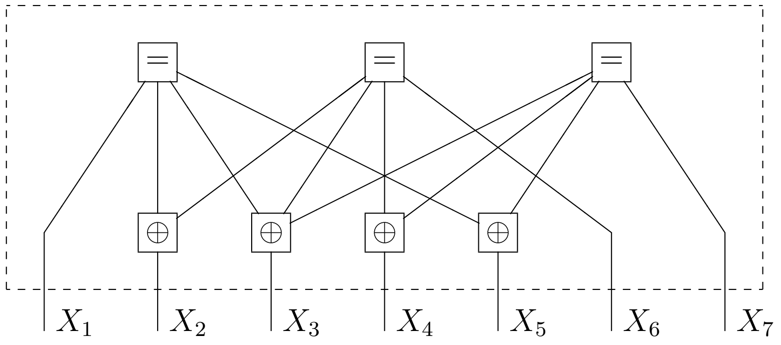


Figure 9: Dualizing Fig. 8 yields an FFG for the dual code.

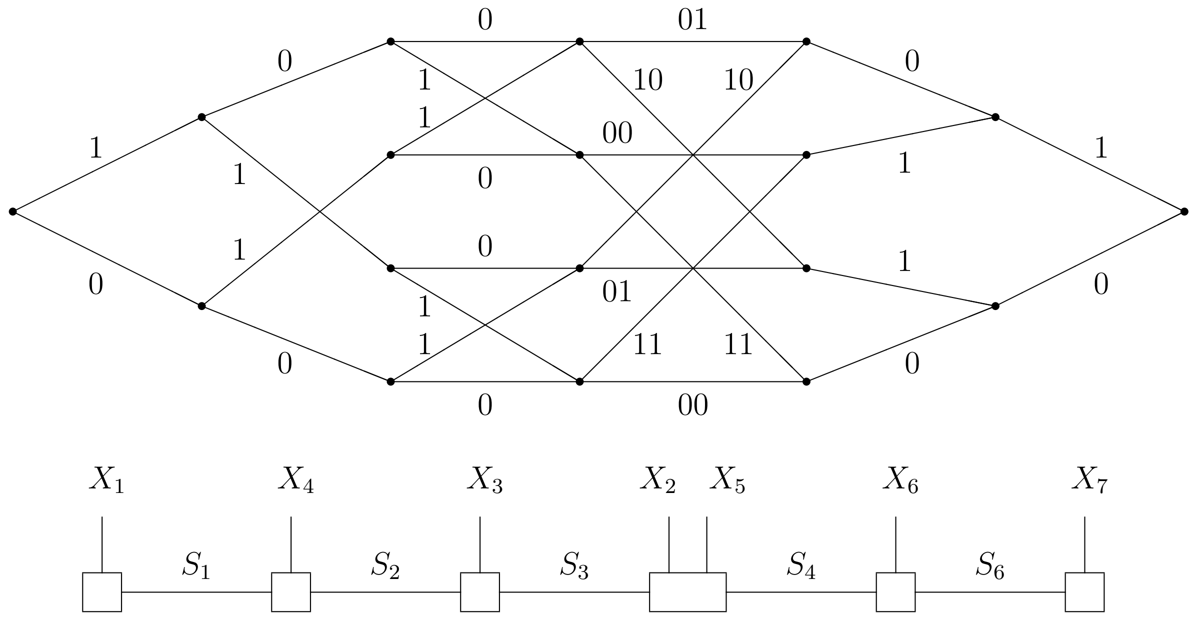


Figure 10: A trellis for the binary $(7,4,3)$ Hamming code (top) and the corresponding FFG (bottom).

As mentioned in the introduction, the standard trellis-based decoding algorithms are instances of the summary product algorithm, which works on any factor graph. In particular, when applied to a trellis, the sum-product algorithm becomes the BCJR algorithm [5] and the max-product algorithm (or the min-sum algorithm applied in the logarithmic domain) becomes a soft-output version of the Viterbi algorithm [10].

The FFG of a general LDPC code is shown in Fig. 11. As in Fig. 8, this FFG corresponds to a parity check matrix. The block length n is typically large; for $n < 1000$, LDPC codes do not work very well. The defining property of an LDPC code is that the parity check matrix is sparse: each parity check node is connected only to a small number of equality constraint nodes, and vice versa. Usually, these connections are "random". The standard decoding algorithm for LDPC codes is the sum-product algorithm; the max-product algorithm as well as various approximations of these algorithms are also sometimes used. More about LDPC codes may be found in [1] and [3]; see also [36] and [40].

The FFG of a generic turbo code is shown in Fig. 12. It consists of two trellises, which share a number of common symbols via a "random" interleaver. Again, the standard decoding algorithm is the sum-product algorithm, with alternating forward-backward (BCJR) sweeps through the two trellises.

Other classes of codes that work well with iterative decoding such as repeat-accumulate codes [8] and zigzag codes [37] have factor graphs similar to those of Fig. 11 and Fig. 12.

A *channel model* is a family $p(y_1, \ldots, y_n \mid x_1, \ldots, x_n)$ of probability distributions over a block $y = (y_1, \ldots, y_n)$ of channel output symbols given any block $x = (x_1, \ldots, x_n)$ of channel input symbols. Connecting, as shown in Fig. 13, the factor graph (Tanner graph) of a code C with the factor graph of a channel model $p(y|x)$ results in a factor graph of the joint likelihood function $p(y|x)I_C(x)$. If we assume that the codewords are equally likely to be transmitted, we have for any fixed received block y

$$\begin{aligned} p(x|y) &= \frac{p(y|x)p(x)}{p(y)} && (19) \\ &\propto p(y|x)I_C(x). && (20) \end{aligned}$$

The joint code/channel factor graph thus represents the *a posteriori* joint probability of the coded symbols $X_1, \ldots, X_n$.

Two examples of channel models are shown in Fig. 14 and Fig. 15. Fig. 14 shows a memoryless channel with

$$p(y|x) = \prod_{k=1}^{n} p(y_k|x_k). \qquad (21)$$

Fig. 15 shows a state space representation with internal states $S_0, S_1, \ldots, S_n$:

$$p(y, s \mid x) = p(s_0) \prod_{k=1}^{n} p(y_k, s_k \mid x_k, s_{k-1}). \qquad (22)$$

Such a state space representation might be, e.g., a finite-state trellis or a linear model as in Fig. 7.

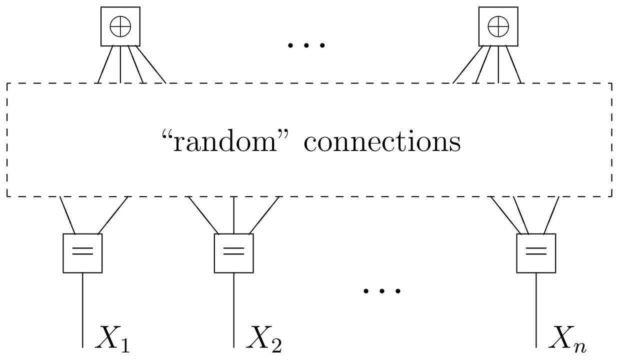


Figure 11: FFG of a low-density parity check code.

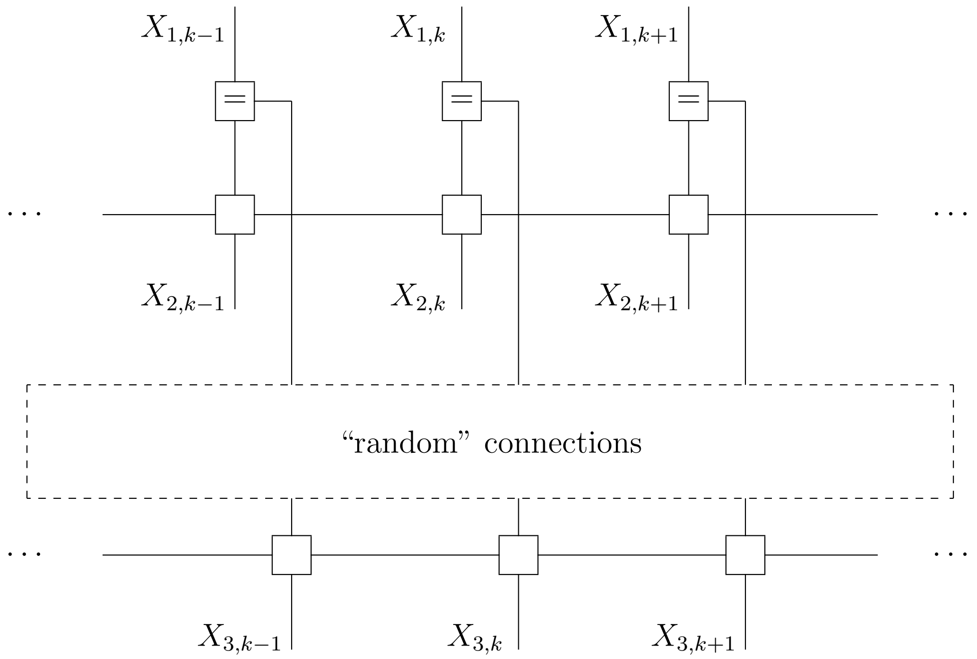


Figure 12: FFG of a parallel concatenated code (turbo code).

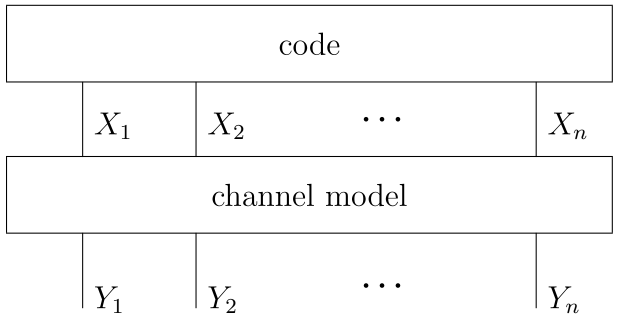


Figure 13: Joint code/channel FFG.

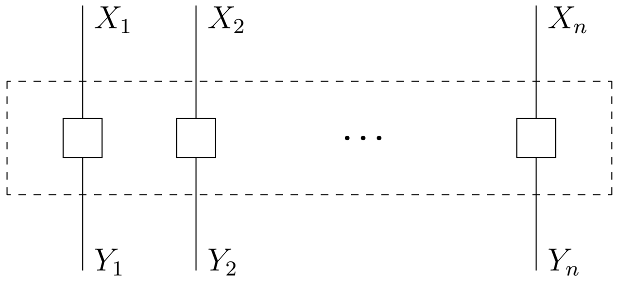


Figure 14: Memoryless channel.

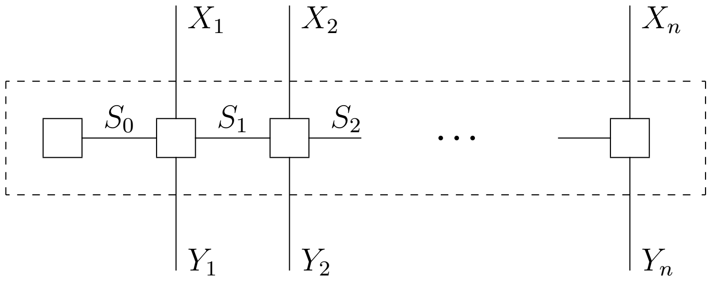


Figure 15: State space channel model.

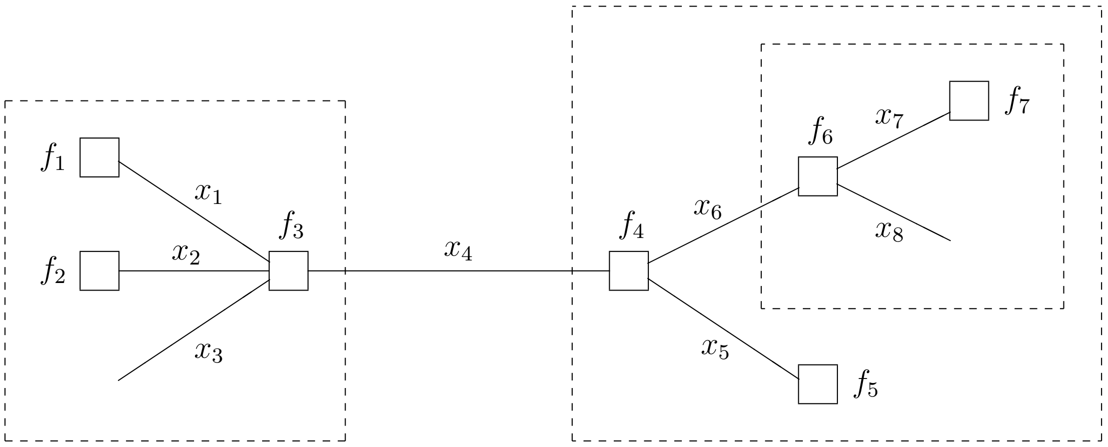


Figure 16: Elimination of variables: "closing the box" around subsystems.

4 Summary Propagation Algorithms

4.1 Closing Boxes: the Sum-Product Rule

So far, we have freely introduced auxiliary variables (state variables) in order to obtain nicely structured models. Now we will consider the elimination of variables. For example, for some discrete probability mass function $f(x_1, \ldots, x_8)$, we might be interested in the marginal probability

$$p(x_4) = \sum_{x_1,x_2,x_3,x_5,x_6,x_7} f(x_1, \ldots, x_8). \tag{23}$$

Or, for some nonnegative function $f(x_1, \ldots, x_8)$, we might be interested in

$$\rho(x_4) \triangleq \max_{x_1,x_2,x_3,x_5,x_6,x_7} f(x_1, \ldots, x_8). \tag{24}$$

The general idea is to get rid of some variables by some "summary operator", and the most popular summary operators are summation (or integration) and maximization (or minimization). Note that only the valid configurations contribute to a sum as in (23), and (assuming that f is nonnegative) only the valid configurations contribute to a maximization as in (24).

Now assume that f has an FFG as in Fig. 16, i.e., f can be written as

$$\begin{aligned} f(x_1, \ldots, x_8) \;\; = \;\; & \Big(f_1(x_1)f_2(x_2)f_3(x_1,x_2,x_3,x_4)\Big) \cdot \\ & \Big(f_4(x_4,x_5,x_6)f_5(x_5)\Big(f_6(x_6,x_7,x_8)f_7(x_7)\Big)\Big). \end{aligned} \tag{25}$$

Note that the brackets in (25) correspond to the dashed boxes in Fig. 16.

Inserting (25) into (23) and applying the distributive law yields

$$
\begin{aligned}
p(x_4) \;=\; & \underbrace{\left(\sum_{x_1} \sum_{x_2} \sum_{x_3} f_3(x_1, x_2, x_3, x_4) f_1(x_1) f_2(x_2) \right)}_{\mu_{f_3 \to x_4}} \cdot \\
& \underbrace{\left(\sum_{x_5} \sum_{x_6} f_4(x_4, x_5, x_6) f_5(x_5) \underbrace{\left(\sum_{x_7} \sum_{x_8} f_6(x_6, x_7, x_8) f_7(x_7) \right)}_{\mu_{f_6 \to x_6}} \right)}_{\mu_{f_4 \to x_4}} \qquad (26)
\end{aligned}
$$

This expression can be interpreted as "closing" the dashed boxes in Fig. 16 by summarizing over their internal variables. The factor $\mu_{f_3 \to x_4}$ is the summary of the big dashed box left in Fig. 16; it is a function of x_4 only. The factor $\mu_{f_6 \to x_6}$ is the summary of the small dashed box right in Fig. 16; it is a function of x_6 only. Finally, the factor $\mu_{f_4 \to x_4}$ is the summary of the big dashed box right in Fig. 16; it is a function of x_4 only. The resulting expression

$$
p(x_4) = \mu_{f_3 \to x_4}(x_4) \cdot \mu_{f_4 \to x_4}(x_4) \qquad (27)
$$

corresponds to the FFG of Fig. 16 with the dashed boxes closed.

Replacing all sums in (26) by maximizations yields an analogous decomposition of (24). In general, it is easy to prove the following fact.

Local Elimination Property: A "global" summary (by summation/integration or by maximization) may be obtained by successive "local" summaries of subsystems.

It is now but a small step to the summary product algorithm. Towards this end, we consider the summaries (i.e., the terms in brackets in (26)) as "messages" that are sent out of the corresponding box, as is illustrated in Fig. 17. We also define the message out of a terminal node (e.g., f_1) as the corresponding function itself (e.g., $f_1(x_1)$). "Open" half edges (such as x_3) do not carry a message towards the (single) node attached to them; alternatively, they may be thought of as carrying as message a neutral factor 1. It is then easy to verify that *all* summaries/messages in Fig. 17 are formed according to the following general rule.

Sum-Product Rule (see Fig. 18): The message out of some node $g(x, y_1, \ldots, y_n)$ along the branch x is the function

$$
\mu_{g \to x}(x) \triangleq \sum_{y_1} \ldots \sum_{y_n} g(x, y_1, \ldots, y_n)\, \mu_{y_1 \to g}(y_1) \cdots \mu_{y_n \to g}(y_n), \qquad (28)
$$

where $\mu_{y_k \to g}$ (which is a function of y_k) is the message that arrives at g along the edge y_k.

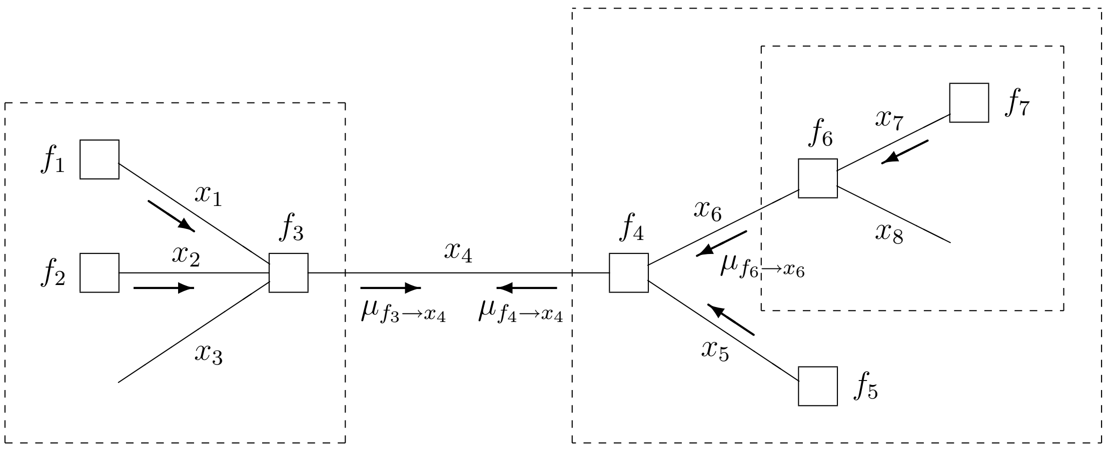


Figure 17: "Summarized" factors as "messages" in the FFG.

If we use maximization as the summary operator, we have the analogous

Max-Product Rule (see Fig. 18): The message out of some node $g(x, y_1, \ldots, y_n)$ along the branch x is the function

$$\mu_{g\to x}(x) \triangleq \max_{y_1} \ldots \max_{y_n} g(x, y_1, \ldots, y_n)\, \mu_{y_1\to g}(y_1) \cdots \mu_{y_n\to g}(y_n). \tag{29}$$

These two rules are instances of the following single rule.

Summary-Product Rule: The message out of a factor node $g(x, \ldots)$ along the edge x is the product of $g(x, \ldots)$ and all messages towards g along all edges except x, summarized over all variables except x.

We have thus seen that

1. Summaries/marginals such as (23) and (24) can be computed as the product of two messages as in (27).
2. Such messages are summaries of the subsystem "behind" them.
3. All messages (except those out of terminal nodes) are computed from other messages according to the summary-product rule.

It is easy to see that this procedure to compute summaries is not restricted to the example of Fig. 16 but applies whenever the factor graph has no cycles.

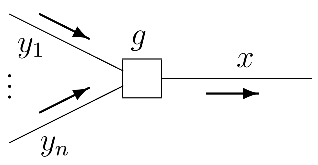


Figure 18: Messages along a generic edge.

4.2 The Summary-Product Algorithm

In its general form, the summary-product algorithm computes two messages for each edge in the graph, one in each direction. Each message is computed according to the summary-product rule (typically the sum-product rule (28) or the max-product rule (29)).

A sharp distinction divides graphs with cycles from graphs without cycles. If the graph has no cycles, then it is efficient to begin the message computation from the leaves and to successively compute messages as their required "input" messages become available. In this way, each message is computed exactly once. It is then obvious from the previous section that summaries/marginals as in (23) or (24) can be computed as the product of messages as in (27) *simultaneously for all variables.*

A simple numerical example is worked out in the box on page 19. Fig. (a) of that example shows an FFG of a toy code, a binary linear code of length $n = 4$ and dimension $k = 2$. In Fig. (b), the FFG is extended to a joint code/channel model as in Fig. 13. The channel output symbols Y_ℓ are binary, and the four nodes in the channel model represent the factors

$$p(y_\ell|x_\ell) = \begin{cases} 0.9, & \text{if } y_\ell = x_\ell \\ 0.1, & \text{if } y_\ell \neq x_\ell \end{cases} \tag{30}$$

for $\ell = 1, \ldots, 4$. If $(Y_1, \ldots, Y_4) = (y_1, \ldots, y_4)$ is known (fixed), the factor graph in Fig. (b) represents the *a posteriori* probability $p(x_1, \ldots, x_4|y_1, \ldots, y_4)$, up to a scale factor, cf. (20).

Figures (c)–(e) of the example show the messages as computed according to the sum-product rule (28). (The message computations for such nodes are given in Table 1.) The final result is the per-symbol *a posteriori* probability $p(x_\ell|y_1, \ldots, y_4)$ for $\ell = 1, \ldots, 4$; according to (27), this is obtained as (a suitably scaled version of) the product of the two messages along the edge X_ℓ.

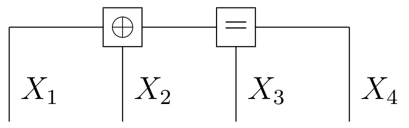
If a trellis code as in Fig. 10 is used with a memoryless channel as in Fig. 14, the overall factor graph as in Fig. 13 (which represents the joint *a posteriori* probability $p(x|y)$, up to a scale factor) has no cycles. For such codes, the natural schedule for the message computations consists of two independent recursions through the trellis, one (forward) from left to right and the other (backward) from right to left. If the sum-product rule is used, this procedure is identical with the BCJR algorithm [5], and we can obtain the *a posteriori* marginal probability $p(b|y)$ of every branch b of the trellis (and hence of every information bit). If the max-product rule is used, the forward recursion is essentially identical with the Viterbi algorithm, except that no paths are stored, but all messages

Sum-Product (Belief Propagation) Algorithm: An Example

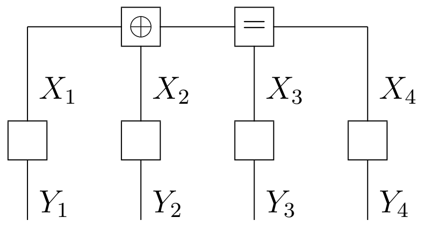
Consider a simple binary code

$$C = \big\{(0,0,0,0),(0,1,1,1),(1,0,1,1),(1,1,0,0)\big\},$$

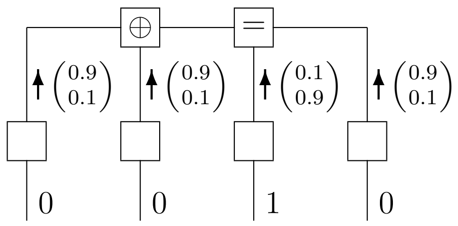
which is represented by the FFG in Fig. (a) below. Assume that a codeword $(X_1,\ldots,X_4)$ is transmitted over a binary symmetric channel with crossover probability $\varepsilon = 0.1$ and assume that $(Y_1,\ldots,Y_4) = (0,0,1,0)$ is received. The figures below show the messages of the sum-product algorithm. The messages μ are represented as $\binom{\mu(0)}{\mu(1)}$, scaled such that $\mu(0)+\mu(1)=1$. The final result in Fig. (f) is the *a posteriori* probability $p(x_\ell|y_1,\ldots,y_4)$ for $\ell = 1,\ldots,4$.



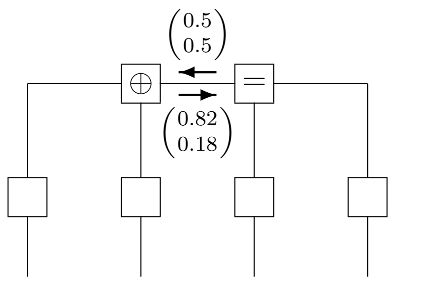
a) FFG of the code.



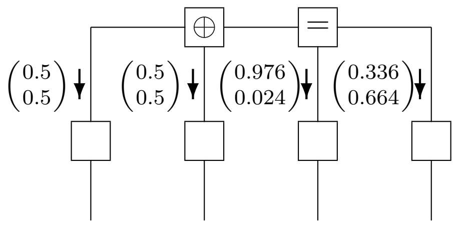
b) Code/channel model.



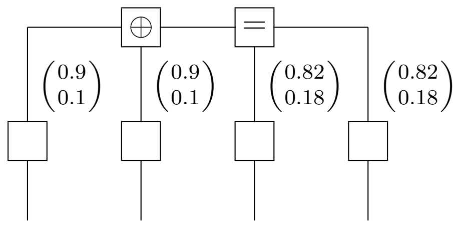
c) Computing messages…



d) Computing messages…



e) Computing messages…



(f) *A posteriori* probabilities obtained from (c) and (e).

(branch metrics) must be stored; the backward recursion is formally identical with the forward recursion; and we can obtain the quantity $\rho(b|y) \stackrel{\triangle}{=} \max_{\omega \in \Omega:\, b \text{ fixed}} p(\omega|y)$ for every branch b of the trellis (and hence for every information bit). As pointed out in [45], the max-product algorithm may thus be viewed as a soft-output Viterbi algorithm, and the Viterbi-algorithm [10] itself may be viewed as an efficient hard-decision-only version of the max-product algorithm.

If the factor graph has cycles, we obtain iterative algorithms. First, all edges are initialized with a neutral message, i.e., a factor $\mu(.) = 1$. All messages are then repeatedly updated, according to some schedule. The computation stops when the available time is over or when some other stopping condition is satisfied (e.g., when a valid codeword was found).

We still compute the final result as in (27), but on graphs with cycles, this result will usually be only an approximation of the true summary/marginal. Nevertheless, when applied to turbo codes as in Fig. 12 or LDPC codes as in Fig. 11, reliable performance very near the Shannon capacity of the channel can be achieved!

If the rule (28) (or (29)) is implemented literally, the values of the messages/functions $\mu(.)$ typically tend quickly to zero (or sometimes to infinity). In practice, therefore, the messages often need to be scaled or normalized (as was done in the example on page 19): instead of the message $\mu(.)$, a modified message

$$\mu'(.) \stackrel{\triangle}{=} \gamma \mu(.) \tag{31}$$

is computed, where the scale factor γ may be chosen freely for every message. The final result (27) will then be known only up to a scale factor, which is usually no problem.

Table 1 shows the sum-product update rule (28) for the building blocks of low-density parity check codes (see Fig. 11). It is quite popular to write these messages in terms of the single parameters

$$L_X \stackrel{\triangle}{=} \log \frac{\mu_X(0)}{\mu_X(1)}, \tag{32}$$

or $\Delta \stackrel{\triangle}{=} \big(\mu(0) - \mu(1)\big) / \big(\mu(0) + \mu(1)\big)$, and the corresponding versions of the update rules are also given in Table 1. Table 2 shows the max-product rules.

For the decoding of LDPC codes the typical update schedule alternates between updating the messages out of equality constraint nodes and updating the messages out of parity check nodes.

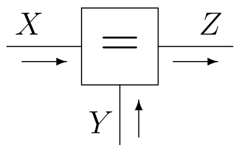
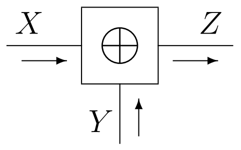
 $\delta(x-y)\,\delta(x-z)$	$\begin{pmatrix}\mu_Z(0)\\ \mu_Z(1)\end{pmatrix} = \begin{pmatrix}\mu_X(0)\,\mu_Y(0)\\ \mu_X(1)\,\mu_Y(1)\end{pmatrix}$ $\Delta_Z = \dfrac{\Delta_X+\Delta_Y}{1+\Delta_X\Delta_Y}$ $L_Z = L_X + L_Y$
 $\delta(x\oplus y\oplus z)$	$\begin{pmatrix}\mu_Z(0)\\ \mu_Z(1)\end{pmatrix} = \begin{pmatrix}\mu_X(0)\,\mu_Y(0)+\mu_X(1)\,\mu_Y(1)\\ \mu_X(0)\,\mu_Y(1)+\mu_X(1)\,\mu_Y(0)\end{pmatrix}$ $\Delta_Z = \Delta_X\cdot\Delta_Y$ $\tanh(L_Z/2) = \tanh(L_X/2)\cdot\tanh(L_Y/2)$

Table 1: Sum-product message update rules for binary parity-check codes.

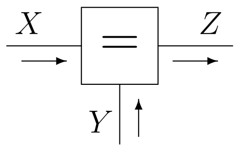
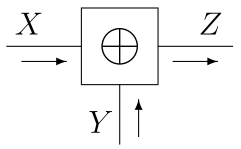
 $\delta(x-y)\,\delta(x-z)$	$\begin{pmatrix}\mu_Z(0)\\ \mu_Z(1)\end{pmatrix} = \begin{pmatrix}\mu_X(0)\,\mu_Y(0)\\ \mu_X(1)\,\mu_Y(1)\end{pmatrix}$ $L_Z = L_X + L_Y$
 $\delta(x\oplus y\oplus z)$	$\begin{pmatrix}\mu_Z(0)\\ \mu_Z(1)\end{pmatrix} = \begin{pmatrix}\max\big\{\mu_X(0)\,\mu_Y(0),\ \mu_X(1)\,\mu_Y(1)\big\}\\ \max\big\{\mu_X(0)\,\mu_Y(1),\ \mu_X(1)\,\mu_Y(0)\big\}\end{pmatrix}$ $\lvert L_Z\rvert = \min\big\{\lvert L_X\rvert,\lvert L_Y\rvert\big\}$ $\operatorname{sgn}(L_Z) = \operatorname{sgn}(L_X)\cdot\operatorname{sgn}(L_Y)$

Table 2: Max-product message update rules for binary parity-check codes.

4.3 Kalman Filtering

An important standard form of the sum-product algorithm is Kalman filtering and smoothing, which amounts to applying the algorithm to the state space model of Fig. 6 and 7 [27], [23]. In the traditional setup, it is assumed that $Y[.]$ is observed and that both $U[.]$ and $W[.]$ are white Gaussian noise. In its most narrow sense, Kalman filtering is then only the forward sum-product recursion through the graph of Fig. 6 (cf. Fig. 19 left) and yields the *a posteriori* probability distribution of the state $X[k]$ given the observation sequence $Y[.]$ up to time k. By computing also the backwards messages (cf. Fig. 19 right), the *a posteriori* probability of all quantities given the whole observation sequence $Y[.]$ may be obtained.

More generally, Kalman filtering amounts to the sum-product algorithm on any factor graph (or part of a factor graph) that consists of Gaussian factors and the linear building blocks listed in Table 3. (It is remarkable that the sum-product algorithm and the max-product algorithm coincide for such graphs.) All messages represent Gaussian distributions. For the actual computation, each such message consists of a mean vector m and a nonnegative definite "cost" matrix (or "potential" matrix) W or its inverse, a covariance matrix $V = W^{-1}$.

A set of rules for the computation of such messages is given in Table 3. As only one of the two messages along any edge, say X, is considered, the corresponding mean vectors and matrices are simply denoted m_X, W_X, etc.

In general, the matrices W and V are only required to be nonnegative definite, which allows to express certainty in V and complete ignorance in W. However, whenever such a matrix needs to be inverted, it had better be positive definite.

The direct application of the update rules in Table 3 may lead to frequent matrix inversions. A key observation in Kalman filtering is that the inversion of large matrices can often be avoided. In the factor graph, such simplifications may be achieved by using the update rules for the composite blocks given in Table 4. (These rules may be derived from those of Table 3 by means of the Matrix Inversion Lemma [19].) In particular, the vectors $U[k]$ and $Z[k]$ in Fig. 6 have usually much smaller dimensions than the state vector $X[k]$; in fact, they are often scalars. By working with composite blocks as in Fig. 19, the forward recursion (left in Fig. 19) using the covariance matrix $V = W^{-1}$ then requires no inversion of a large matrix and the backward recursion (right in Fig. 19) using the cost matrix W requires only one such inversion for each discrete time index.

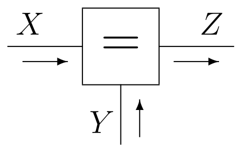
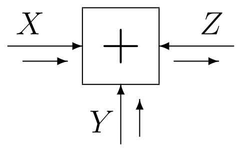
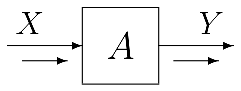
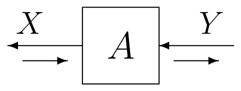
1	 $\delta(x-y)\delta(x-z)$	$m_Z = \big(W_X + W_Y\big)^{\#}\,(W_X m_X + W_Y m_Y)$ $W_Z = W_X + W_Y$ $V_Z = V_X\big(V_X + V_Y\big)^{\#}\,V_Y$
2	 $\delta(x+y+z)$	$m_Z = -m_X - m_Y$ $V_Z = V_X + V_Y$ $W_Z = W_X\big(W_X + W_Y\big)^{\#}\,W_Y$
3	 $\delta(y-Ax)$	$m_Y = A m_X$ $V_Y = A V_X A^H$
4	 $\delta(x-Ay)$	$m_Y = \big(A^H W_X A\big)^{\#} A^H W_X m_X$ $W_Y = A^H W_X A$ If A has full row rank: $m_Y = A^H\big(AA^H\big)^{-1} m_X$

Table 3: Update rules for messages consisting of mean vector m and covariance matrix V or $W = V^{-1}$. Notation: $(.)^H$ denotes Hermitian transposition and $(.)^{\#}$ denotes the Moore-Penrose pseudo-inverse.

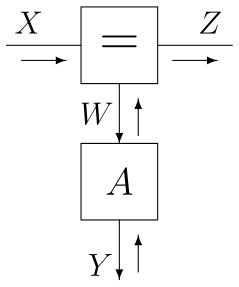
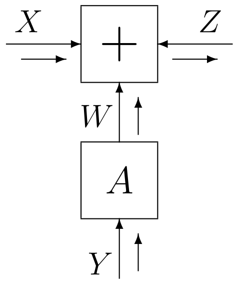
5		$m_Z = m_X + V_X A^H G \left(m_Y - A m_X \right)$ $V_Z = V_X - V_X A^H G A V_X$ with $G \overset{\triangle}{=} \left(V_Y + A V_X A^H \right)^{-1}$
6		$m_Z = -m_X - A m_Y$ $W_Z = W_X - W_X A H A^H W_X$ with $H \overset{\triangle}{=} \left(W_Y + A^H W_X A \right)^{-1}$

Table 4: Update rules for composite blocks.

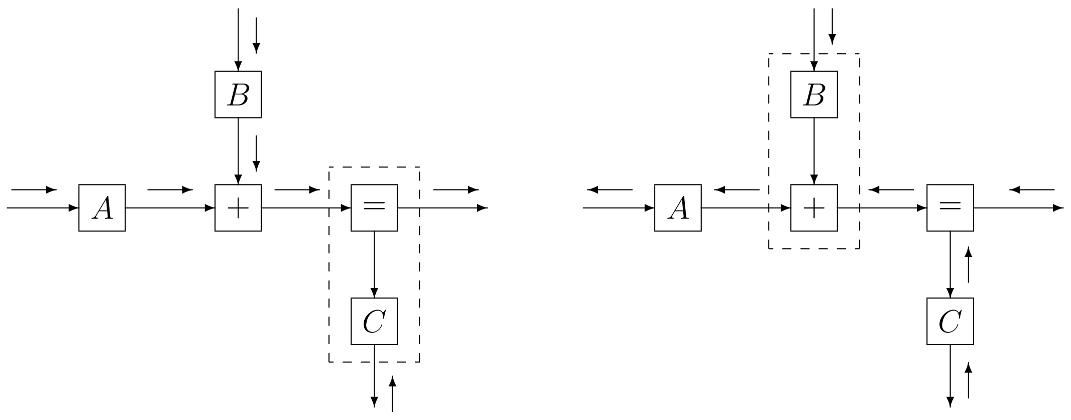


Figure 19: Use of composite-block update rules of Table 4.

4.4 Designing New Algorithms

Factor graphs can be used to model complex real-world systems and to derive practical message passing algorithms for the associated detection and estimation problems. A key issue in most such applications is the coexistence of discrete and continuous variables; another is the harmonic cooperation of a variety of different signal processing techniques. The following design choices must be addressed in any such application.

Choice of the factor graph. In general, the graph (i.e., the equation system that defines the model) is far from unique, and the choice affects the performance of message passing algorithms.

Choice of message types and the corresponding **update rules** for continuous variables (see below).

Scheduling of the message computations.

Discrete variables can usually be handled by literal application of the sum-product (or max-product) rule, or by some obvious approximation of it. For continous variables, literal application of the sum-product or max-product update rules often leads to intractable integrals. Dealing with continuous variables thus involves the choice of suitable message types and of the corresponding (exact or approximate) update rules. The following message types have proved useful.

Constant messages. The message is a "hard-decision" estimate of the variable. Using this message type amounts to inserting decisions on the corresponding variables (as, e.g., in a decision-feedback equalizer).

Quantized messages are an obvious choice. However, quantization is usually infeasible in higher dimensions.

Mean and variance of (exact or assumed) **Gaussian Messages.** This is the realm of Kalman filtering.

The derivative of the message at a single point is the data type used for gradient methods [31].

List of samples. A probability distribution can be represented by a list of samples ("particles") from the distribution. This data type is the basis of the *particle filter* [9]; its use for message passing algorithms in general graphs seems to be largely unexplored, but promising.

Compound messages consist of a combination (or "product") of other message types.

Note that such messages are still summaries of everything "behind" them. With these message types, it is possible to integrate most good known signal processing techniques into summary propagation algorithms for factor graphs.

5 A Glimpse at Some Further Topics

Convergence of Message Passing on Gaussian Graphs. In general, little is known about the performance of message passing algorithms on graphs with cycles. However, in the important special case where the graph represents a *Gaussian* distribution of many variables, Weiss and Freeman [44] and Rusmevichientong and Van Roy [39] have proved the following: *If* sum-product message passing (probability propagation) converges, then the calculated *means* are correct (but the variances are optimistic).

Improved Message Passing on Graphs with Cycles. On graphs with many short cycles, sum-product message passing as described does usually not work well. Some improved (and more complex) message passing algorithms have recently been proposed for such cases, see [51], [35]. A related idea is to use messages with some nontrivial internal Markov structure [7].

Factor Graphs and Analog VLSI Decoders. As observed in [28] (see also [20]), factor graphs for codes (such as Fig. 8, Fig. 9, and Fig. 10) can be directly translated into analog transistor circuits that perform sum-product message passing in parallel and in continuous time. These circuits appear so natural that one is tempted to conjecture that transistors prefer to compute with probabilities!

Such analog decoders may become useful when very high speed or very low power consumption are required, or they might enable entirely new applications of coding. A light introduction to such analog decoders was given in [29]. More extensive accounts are [21], [30], [32]. For some recent progress reports, see [48], [16].

Gaussian Factor Graphs and Static Electrical Networks. Static electrical networks consisting of voltage sources, current sources, and resistors are isomorphic with the factor graphs of certain Gaussian distributions. The electrical network "solves" the corresponding least-squares (or Kalman filtering) problem [42], [43].

Fourier Transform and Duality. Forney [13] and Mao and Kschischang [34] showed that the Fourier transform of a multi-variable function can be carried out directly in the FFG (which may have cycles) according to the following recipe:

- Replace each variable by its dual ("frequency") variable.
- Replace each local function by its Fourier transform. If some local function is the membership indicator function $\delta_V(.)$ of a vector space V, its "Fourier transform" is the membership indicator function $\delta_{V^\perp}(.)$ of the orthogonal complement $V^\perp$.
- For each edge, introduce a minus sign into one of the two adjacent factors.

For this recipe to work, all variables of interest must be *external*, i.e., represented by half edges. For example, Fig. 20 illustrates the familiar fact that the Fourier

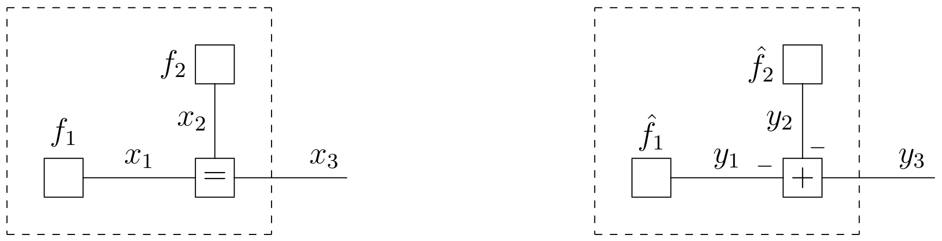


Figure 20: The Fourier transform of pointwise multiplication (left) is convolution (right).

transform of the pointwise multiplication

$$\begin{aligned} f(x_3) &= \sum_{x_1,x_2} f_1(x_1)f_2(x_2)\delta(x_1-x_3)\delta(x_2-x_3) && (33)\\ &= f_1(x_3)f_2(x_3) && (34)\end{aligned}$$

is the convolution

$$\begin{aligned} \hat{f}(y_3) &= \sum_{y_1,y_2} \hat{f}_1(y_1)\hat{f}_2(y_2)\delta(y_3-y_2-y_1) && (35)\\ &= \sum_{y_2} \hat{f}_1(y_3-y_2)\hat{f}_2(y_2). && (36)\end{aligned}$$

6 Conclusion

Graphical models such as factor graphs allow a unified approach to a number of key topics in coding and signal processing: the iterative decoding of turbo codes, low-density parity check codes, and similar codes; joint decoding and equalization; joint decoding and parameter estimation; hidden-Markov models; Kalman filtering and recursive least squares, and more. Graphical models can represent complex real-world systems, and such representations help to derive practical detection/estimation algorithms in a wide area of applications. Most good known signal processing techniques—including gradient methods, Kalman filtering, and particle methods—can be used as components of such algorithms. Other than most of the previous literature, we have used Forney-style factor graphs, which support hierarchical modeling and are compatible with standard block diagrams.

References

[1] Special issue on "Codes and graphs and iterative algorithms" of *IEEE Trans. Inform. Theory,* vol. 47, Feb. 2001.

[2] Special issue on "The turbo principle: from theory to practice II" of *IEEE J. Select. Areas Comm.,* vol. 19, Sept. 2001.

[3] Collection of paper on "Capacity approaching codes, iterative decoding algorithms, and their applications" in *IEEE Communications Mag.,* vol. 41, August 2003.

[4] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory,* vol. 46, pp. 325–343, March 2000.

[5] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory,* vol. 20, pp. 284–287, March 1974.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon-limit error-correcting coding and decoding: Turbo codes," *Proc. 1993 IEEE Int. Conf. Commun.,* pp. 1064–1070, Geneva, May 1993.

[7] J. Dauwels, H.-A. Loeliger, P. Merkli, and M. Ostojic, "On structured-summary propagation, LFSR synchronization, and low-complexity trellis decoding," *Proc. 41st Allerton Conf. on Communication, Control, and Computing,* Monticello, Illinois, October 1-3, 2003, to appear.

[8] D. Divsalar, H. Jin, and R. J. McEliece, "Coding theorems for 'turbo-like' codes," *Proc. 36th Allerton Conf. on Communication, Control, and Computing,* Allerton, Ill., Sept. 1998, pp. 201–210.

[9] Djuric et al., "Particle filtering," *IEEE Signal Proc. Mag.,* September 2003, pp. 19–38.

[10] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE,* vol. 61, pp. 268–278, March 1973.

[11] G. D. Forney, Jr., "On iterative decoding and the two-way algorithm," *Proc. Int. Symp. on Turbo Codes and Related Topics*, Brest, France, Sept. 1997.

[12] G. D. Forney, Jr., "Codes on graphs: news and views," *Proc. Int. Symp. on Turbo Codes and Related Topics,* Sept. 4–7, 2000, Brest, France, pp. 9–16.

[13] G. D. Forney, Jr., "Codes on graphs: normal realizations," *IEEE Trans. Inform. Theory,* vol. 47, no. 2, pp. 520–548, 2001.

[14] B. J. Frey and F. R. Kschischang, “Probability propagation and iterative decoding,” *Proc. 34th Annual Allerton Conf. on Commun., Control, and Computing*, (Allerton House, Monticello, Illinois), Oct. 1–4, 1996.

[15] B. J. Frey, F. R. Kschischang, H.-A. Loeliger, and N. Wiberg, “Factor graphs and algorithms,” *Proc. 35th Allerton Conf. on Communications, Control, and Computing,* (Allerton House, Monticello, Illinois), Sept. 29 – Oct. 1, 1997, pp. 666–680.

[16] M. Frey, H.-A. Loeliger, F. Lustenberger, P. Merkli, and P. Strebel, “Analog-decoder experiments with subthreshold CMOS soft-gates,” *Proc. 2003 IEEE Int. Symp. on Circuits and Systems,* Bangkok, Thailand, May 25–28, 2003, vol. 1, pp. 85-88.

[17] R. G. Gallager, *Low-Density Parity-Check Codes.* Cambridge, MA: M.I.T. Press, 1963.

[18] Z. Ghahramani and M. I. Jordan, “Factorial Hidden Markov models.” *Neural Information Processing Systems,* vol. 8, pp. 472–478, 1995.

[19] G. H. Golub and C. F. Van Loan, *Matrix Computations,* North Oxford Academic, 1986.

[20] J. Hagenauer, “Decoding of binary codes with analog networks,” *Proc. 1998 Information Theory Workshop,* San Diego, CA, Feb. 8–11, 1998, pp. 13–14.

[21] J. Hagenauer, E. Offer, C. Méasson, and M. Mörz, “Decoding and equalization with analog non-linear networks,” *Europ. Trans. Telecomm.,* vol. 10, pp. 659–680, Nov.-Dec. 1999.

[22] M. I. Jordan and T.J. Sejnowski, eds., *Graphical Models: Foundations of Neural Computation.* MIT Press, 2001.

[23] M. I. Jordan, *An Introduction to Probabilistic Graphical Models,* in preparation.

[24] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Trans. Inform. Theory,* vol. 47, pp. 498–519, Feb. 2001.

[25] F. R. Kschischang, “Codes defined on graphs,” *IEEE Signal Proc. Mag.,* vol. 41, August 2003, pp. 118–125.

[26] S. L. Lauritzen and D. J. Spiegelhalter, “Local computations with probabilities on graphical structures and their application to expert systems,” *J. Royal Statistical Society B,* pp. 157–224, 1988.

[27] H.-A. Loeliger, “Least squares and Kalman filtering on Forney graphs,” in *Codes, Graphs, and Systems,* R. E. Blahut and R. Koetter, eds., Kluwer, 2002, pp. 113–135.

[28] H.-A. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarköy, "Probability propagation and decoding in analog VLSI," *Proc. 1998 IEEE Int. Symp. Inform. Th.,* Cambridge, MA, USA, Aug. 16–21, 1998, p. 146.

[29] H.-A. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarköy, "Decoding in analog VLSI," *IEEE Commun. Mag.,* pp. 99–101, April 1999.

[30] H.-A. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarköy, "Probability propagation and decoding in analog VLSI," *IEEE Trans. Inform. Theory,* vol. 47, pp. 837–843, Feb. 2001.

[31] H.-A. Loeliger, "Some remarks on factor graphs," *Proc. 3rd Int. Symp. on Turbo Codes and Related Topics,* Sept. 1–5, 2003, Brest, France, pp. 111–115.

[32] F. Lustenberger *On the Design of Analog VLSI Iterative Decoders.* Diss. ETH No 13879, Nov. 2000.

[33] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory,* vol. 45, pp. 399–431, March 1999.

[34] Y. Mao and F. R. Kschischang, "On factor graphs and the Fourier transform," *Proc. 2001 IEEE Int. Symp. Inform. Th.,* Washington, D.C., USA, June 24–29, 2001, p. 224.

[35] R. J. McEliece and M. Yildirim, "Belief propagation on partially ordered sets," in *Mathematical Systems Theory in Biology, Communication, Computation, and Finance,* J. Rosenthal and D. S. Gilliam, eds., IMA Volumes in Math. and Appl., Springer Verlag, pp. 275–299.

[36] J. Moura, Lu, and Zhang, "Structured LDPC codes with large girth," *IEEE Signal Proc. Mag.,* to appear.

[37] L. Ping, X. Huang, and N. Phamdo, "Zigzag codes and concatenated zigzag codes," *IEEE Trans. Inform. Theory,* vol. 47, pp. 800–807, Feb. 2001.

[38] J. Pearl, *Probabilistic Reasoning in Intelligent Systems,* 2nd ed. San Francisco: Morgan Kaufmann, 1988.

[39] P. Rusmevichientong and B. Van Roy, "An analysis of belief propagation on the turbo decoding graph with Gaussian densities," *IEEE Trans. Inform. Theory,* vol. 47, pp. 745–765, Feb. 2001.

[40] Song and Kumar, "Low-density parity check codes for partial response channels," *IEEE Signal Proc. Mag.,* to appear.

[41] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory,* vol. 27, pp. 533–547, Sept. 1981.

[42] P. O. Vontobel and H.-A. Loeliger, "On factor graphs and electrical networks," in *Mathematical Systems Theory in Biology, Communication, Computation, and Finance,* J. Rosenthal and D. S. Gilliam, eds., IMA Volumes in Math. and Appl., Springer Verlag, pp. 469–492.

[43] P. O. Vontobel, *Kalman Filters, Factor Graphs, and Electrical Networks.* Internal report INT/200202, ISI-ITET, ETH Zurich, April 2002.

[44] Y. Weiss and W. T. Freeman, "On the optimality of the max-product belief propagation algorithm in arbitrary graphs," *IEEE Trans. Inform. Theory* vol. 47, no. 2, pp. 736–744, 2001.

[45] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *Europ. Trans. Telecommunications,* vol. 6, pp. 513–525, Sept/Oct. 1995.

[46] N. Wiberg, *Codes and Decoding on General Graphs.* Linköping Studies in Science and Technology, Ph.D. Thesis No. 440, Univ. Linköping, Sweden, 1996.

[47] G. Winkler, *Image Analysis, Random Fields and Markov Chain Monte Carlo Methods.* 2nd ed., Springer Verlag, 2003.

[48] C. Winstead, J. Dai, W. J. Kim, S. Little, Y.-B. Kim, C. Myers, and C. Schlegel, "Analog MAP decoder for (8,4) Hamming code in subthreshold CMOS," *Proc. Advanced Research in VLSI Conference,* Salt Lake City, Utah, March 2001, pp. 132–147.

[49] R. D. Shachter, "Probabilistic inference and influence diagrams," *Operations Research,* vol. 36, pp. 589–605, 1988.

[50] G. R. Shafer and P. P. Shenoy, "Probability propagation," *Ann. Mat. Art. Intell.,* vol. 2, pp. 327–352, 1990.

[51] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Generalized Belief Propagation," *Advances in Neural Information Processing Systems (NIPS),* vol. 13, pp. 689–695, December 2000.