

Codes Defined on Graphs

Frank R. Kschischang, University of Toronto

ABSTRACT

Low-density parity-check codes, turbo codes, and indeed most practically decodable capacity-approaching error correcting codes can all be understood as codes defined on graphs. Graphs not only describe the codes, but, more important, they structure the operation of the sum-product decoding algorithm (or one of many possible variations), which can be used for iterative decoding. Such coding schemes have the potential to approach channel capacity, while maintaining reasonable decoding complexity. In this tutorial article we review factor graphs, which can be used to describe codes and the joint probability distributions that must be dealt with in decoding. We also review the sum-product algorithm, and show how this algorithm leads to iterative decoding algorithms for codes defined on graphs.

INTRODUCTION

The past decade has been an exciting one in channel coding. Coding theory is motivated by the lure of reliable communication over noisy channels at high code rates; indeed, Claude Shannon's famous Noisy Channel Coding Theorem assures us that codes exist that can, in principle, be decoded reliably at all code rates less than channel capacity. A central challenge in coding theory has always been to devise coding schemes that come close to achieving capacity with practical decoding complexity. Now, following a decade of research sparked by the breakthrough introduction of turbo codes by Berrou *et al.* [1], we have a number of specific "turbo-like" code families (turbo codes, low-density parity-check codes, and others) that, with simple iterative decoding algorithms, achieve performance very close to the Shannon limit on many important channels. The practical application of these codes has not lagged far behind their invention, and we now have turbo-like codes being specified in a wide variety of telecommunications standards and being applied in many communication systems.

A common feature of these capacity-approaching coding schemes is that they all may be understood as *codes defined on graphs*. The purpose of this article is to provide a tutorial introduction to this viewpoint. Now *every* code can be thought of as being defined on a graph, so in a sense the class of codes defined on graphs encompasses all codes. However, practical capacity-approaching turbo-like codes have graphs structured so that on one hand, iterative algorithms such as the sum-product algorithm can operate with relatively small complexity, while on the other hand, the codes themselves may in many cases nearly achieve capacity.

As we will see, graphs provide a means of visualizing the constraints that define the code. More important, however, the graphs directly specify iterative decoding algorithms, which in fact are the algorithms used in practice to decode turbo and other capacity-approaching codes. Certain graph properties (e.g., the degree of the vertices) determine the decoding complexity, while other properties (e.g., girth and diameter) are, at least qualitatively, connected to the performance of the iterative algorithms. Indeed, the best analyses we have of code performance are based on graphical structure. Related communication system components (like sources and channels) can themselves often be modeled with graphs connected to the code graph, which leads to unified receiver designs encompassing such functions as channel estimation, equalization, interference cancellation, and source decoding. The graphical picture also allows connections to be made to similar structures that have arisen in other fields, such as belief propagation in artificial intelligence, Kalman filtering and smoothing, and inference in statistical physics.

This article describes codes defined on graphs in quite general terms. Other articles in this issue (see also the special issue of *IEEE Transactions on Information Theory* [2]) provide more details about the design, analysis, and optimization of specific code families.

GRAPHICAL MODELS FOR CODES

For simplicity, all of the codes considered in this article will be binary linear (n, k) block codes. These are codes in which each codeword is a binary vector of length n , and the set of codewords forms a k -dimensional subspace of the vector space \mathbb{F}_2^n of binary n -tuples. The rate of such a code is k/n . The modulo-two sum of binary variables x and y will be denoted $x \oplus y$. The restriction to binary codes is not essential as all of the concepts discussed here can readily be generalized to nonbinary codes.

TANNER GRAPHS

Graphical models for probability distributions have a rich history. For example, (undirected) Markov random fields and (directed) Bayesian networks are widely used in statistical physics, statistics, and artificial intelligence. In coding theory, codes connected with graphs have been defined in a number of ways. Factor graphs [3], the graphical model used in this article, have their origins in the work of Tanner [4], who defined a bipartite graph — now called a Tanner graph — that interrelates the symbols of a codeword and the constraints these symbols must satisfy in order to be a valid codeword. Tanner also described two graph-based decoding algorithms, to which we refer here as the sum-product and min-sum algorithms.

Figure 1a shows a Tanner graph for the famous $(7, 4)$ Hamming code. There are seven variable vertices, shown as circles, corresponding to the seven symbols x_1, x_2, \dots, x_7 in each codeword. There are also three check vertices, representing the binary linear equations each codeword must satisfy. In a valid codeword, the neighbors of every check vertex (i.e., the variables connected to the check by a single edge) must form a configuration with a binary sum of zero (i.e., a configuration with an even number of ones). Thus, for example, the configurations $(x_1, \dots, x_7) = (1, 1, 1, 0, 0, 0, 0)$ and $(x_1, \dots, x_7) = (0, 1, 1, 1, 0, 1, 0)$ are valid codewords, whereas $(x_1, \dots, x_7) = (1, 1, 1, 0, 1, 0, 1, 0)$ is not a valid codeword, since not every check is satisfied. Notice that this Tanner graph contains cycles (i.e., the graph is not a tree). As we will see, cycle-free graphs support optimal noniterative decoding, whereas graphs with cycles in general support suboptimal iterative decoding.

A Tanner graph representing a code can be obtained from any system of parity check equations that define the code. Since a given code may correspond to more than one such system (i.e., linear codes do not have unique parity-check matrices), it follows that a given code may in general be described by more than one Tanner graph. The low-density parity check (LDPC) codes of Gallager, while not originally described in the language of graph theory, are a prototypical example of codes defined on Tanner graphs. Figure 2 shows the Tanner graph for a very short $(6, 3)$ regular LDPC code (i.e., an LDPC code in which every check vertex has degree 6 and every variable vertex has degree 3). The block labeled Π in Fig. 2 denotes a permutation, a block in

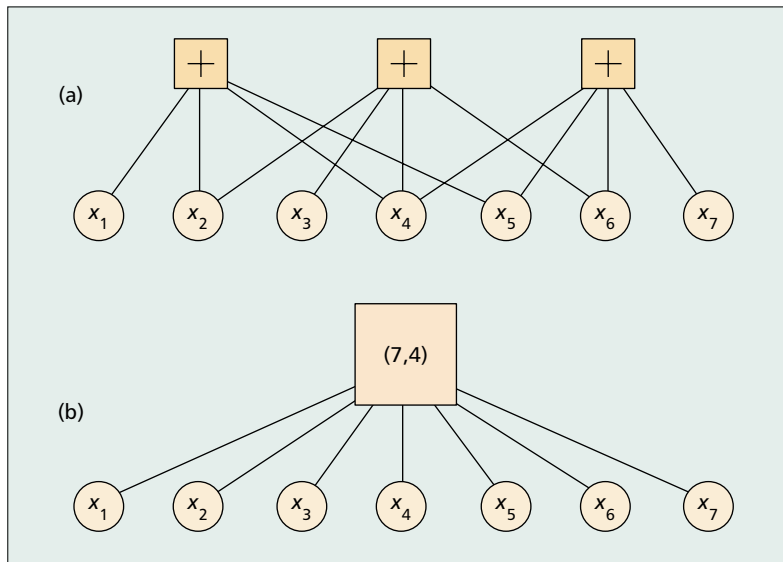


Figure 1. Two Tanner graphs for the $(7,4)$ Hamming code: a) local check version; b) global check version.

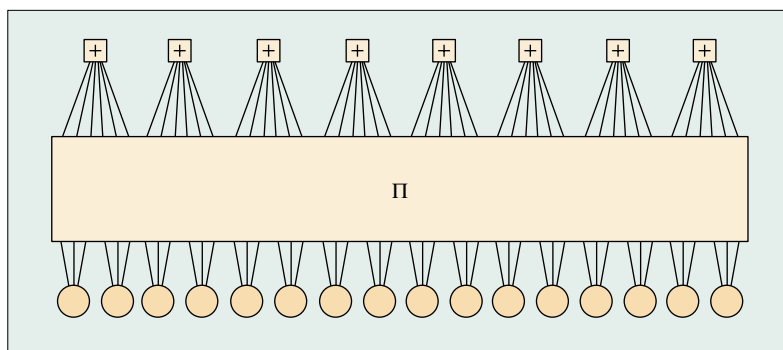


Figure 2. Tanner graph for a very short $(6, 3)$ regular low-density parity check code.

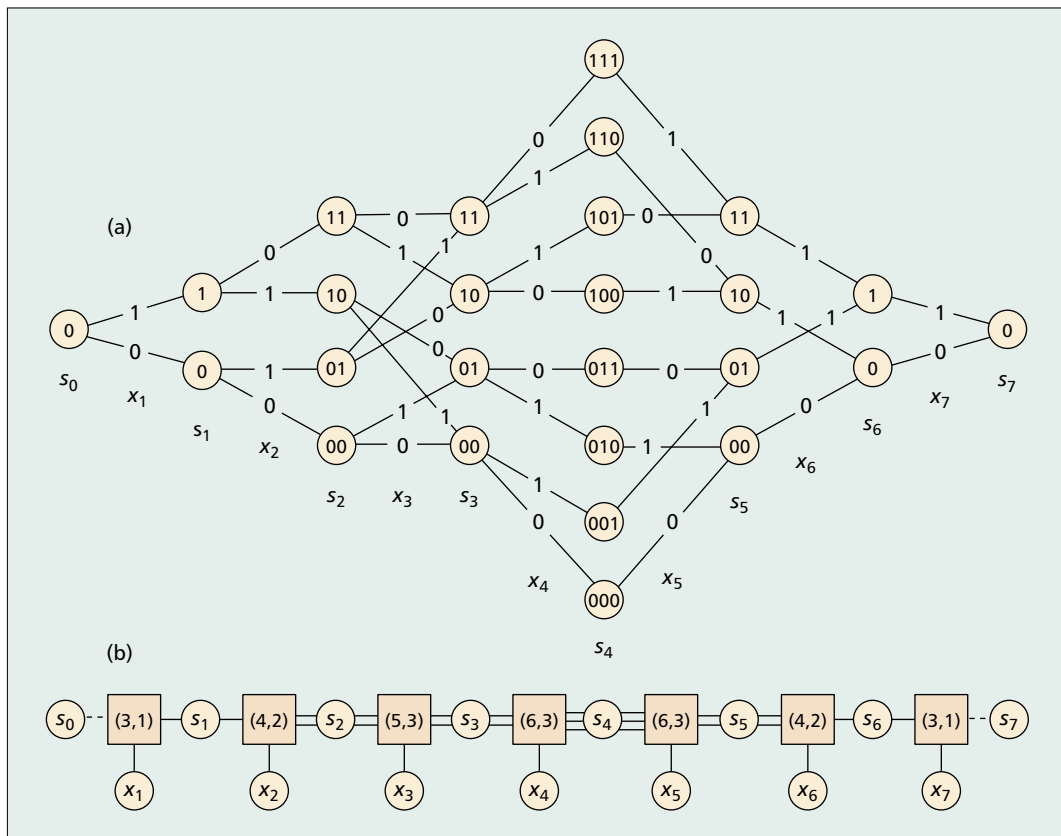
which there is a one-to-one correspondence (in some scrambled order) between the edges incident on one side of the block with those incident on the other side.

The “checks” in a Tanner graph may be generalized to be more complicated than simple parity check equations. For example, the $(7, 4)$ code could also be specified by a Tanner graph with just a single global check vertex that requires that its neighbors form a valid Hamming codeword, as shown in Fig. 1b. In effect, the global check is formed by clustering together several simple checks. Such clustering can convert the graph to a tree or otherwise reduce cycles in the graph, but at the expense of greater check complexity.

FACTOR GRAPHS

Wiberg, Loeliger, and Koetter independently devised Tanner graphs [5, 6], but added the flexibility of auxiliary (state) variables. Such auxiliary variables are not themselves codeword symbols, but their introduction may simplify the graph associated with a code. An example is a trellis, which corresponds to a graph with a particularly simple cycle-free chain structure. We will generally refer to Tanner graphs that may contain state variables as *factor graphs*.

The utility of graph-based modeling of codes becomes evident only when one considers the problem of decoding. Decoding is essentially a problem of statistical inference: given a set of observations, infer which codeword was most likely to have been sent, or infer the most likely value of each individual code symbol.



■ **Figure 3.** a) A trellis diagram for the (7, 4) Hamming code; b) the corresponding factor graph.

For example, Fig. 3a shows a trellis diagram for the (7, 4) Hamming code, and Fig. 3b shows the corresponding factor graph. The trellis diagram represents the 16 codewords of the Hamming code as the set of labeled paths obtained starting from the leftmost vertex and proceeding rightwards in the graph. The auxiliary variables s_0, s_1, \dots, s_7 are not directly code symbols, but their introduction simplifies the factor graph structure, making it cycle-free. The i th trellis section constrains the possible combinations of (s_{i-1}, x_i, s_i) ; in fact, in this linear trellis example, these triples always form a linear code. For example, the third trellis section forms the local code that consists of the eight binary linear combinations of (00, 1, 01), (01, 0, 10), and (10, 0, 01). This code may be regarded as a binary (5, 3) code, and is labeled as such in Fig. 3b. As a reminder that the state variables are in general nonbinary, the factor graph of Fig. 3b shows parallel edges corresponding to the number of bits that make up each state variable. (The endmost states s_0 and s_7 are constrained always to be zero, so in fact they are 0-bit variables or constants, and thus their connection to the factor graph of Fig. 3b is shown with dotted lines. In principle, this connection could be omitted, which would change the local code to which they are now connected into a (2, 1) code.)

Convolutional codes are also commonly described with trellises, and thus also have a factor graph with the chain-like structure shown in Fig. 3b, but typically with the same local code at each point in the chain. The main difficulty with

trellises and other cycle-free graph representations of codes is that as codes become more powerful, the alphabets (state spaces) of the state variables in cycle-free graphs must necessarily become exponentially large [6, Sec. 7.1], which eventually makes trellis-type decoding algorithms impractical.

Factor graphs can be used to model a variety of coding structures. For example, turbo codes are structured according to the encoder diagram shown in Fig. 4a. The encoder takes in k information bits $x = (x_1, \dots, x_k)$, and produces two distinct streams of parity check bits, $\mathbf{p} = (p_1, \dots, p_k)$ and $\mathbf{q} = (q_1, \dots, q_k)$, chosen so that the pairs (\mathbf{x}, \mathbf{p}) and $(\Pi\mathbf{x}, \mathbf{q})$ are codewords in $(2k, k)$ codes C_1 and C_2 , respectively. (Usually, $C_1 = C_2$.) As with LDPC codes, the symbol Π represents a *permutation* of the information bits; \mathbf{q} is computed from the same information bits that determine \mathbf{p} , but taken in a different order. Figure 4b shows a generic factor graph in which the constituent codes are modeled by a single check vertex. In Fig. 4c, these global checks are decomposed into chain graphs that correspond to trellis representations of C_1 and C_2 .

DECODING USING THE SUM-PRODUCT ALGORITHM

The utility of graph-based modeling of codes becomes evident only when one considers the problem of decoding. Decoding is essentially a problem of statistical inference: given a set of

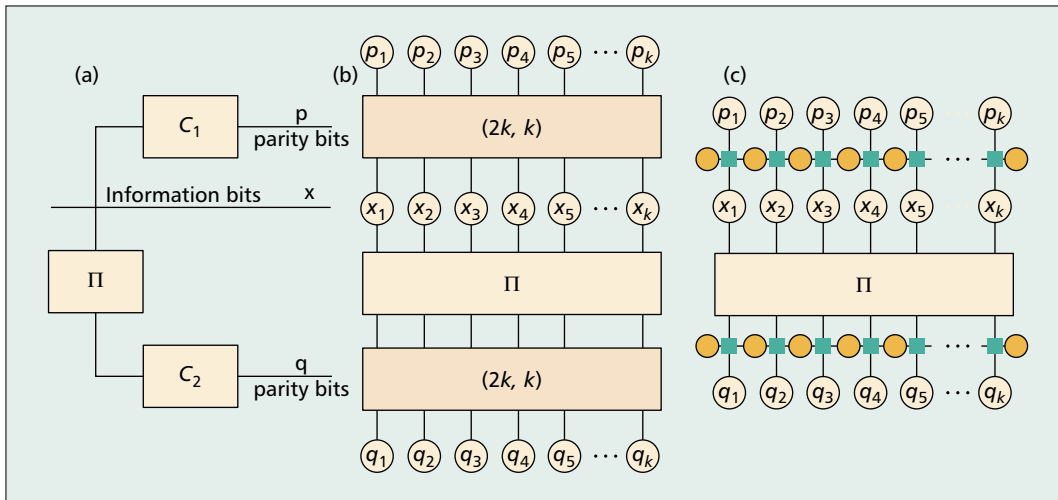


Figure 4. a) The encoder for a parallel concatenated code; b) a generic factor graph; c) factor graph for trellis-structured constituent codes.

observations (the output of a noisy channel in response to the transmission of a codeword), infer which codeword was most likely to have been sent (ML decoding), or the most likely value of each individual code symbol (symbol-by-symbol MAP decoding).

We will describe an algorithm — the sum-product algorithm — that attempts to solve the symbol-by-symbol MAP decoding problem by passing messages along the edges of a factor graph. The well-known belief propagation algorithm used in Bayesian networks is a special case of the sum-product algorithm. The connection between iterative algorithms used in turbo decoding and the belief propagation algorithms used in artificial intelligence was observed in [7, 8].

FACTOR GRAPH REPRESENTATIONS OF PROBABILITY MASS FUNCTIONS

Although we have seen that factor graphs can be used to model codes, these graphs derive their name from the fact that they can, in general, represent the factorization structure of a function of many variables. For example, suppose that the function $g(x_1, \dots, x_7)$ factors as a product of three functions: $f_1(x_1, x_2, x_4, x_5)$, $f_2(x_2, x_3, x_4, x_6)$ and $f_3(x_4, x_5, x_6, x_7)$. A factor graph for g would then have precisely the structure of the Hamming code factor graph shown in Fig. 1a, but with the checks replaced by factor vertices corresponding to f_1, f_2 , and f_3 .

In general, suppose g is some global function with arguments x_1, \dots, x_n , and that g factors as $g = \prod_{j=1}^m f_j$, where each f_j is a local function having some subset of x_1, \dots, x_n as arguments. A factor graph representing this factorization of g has n variable vertices (one for each argument of g) and m factor vertices (one for each factor f_j of g). An edge connects a variable vertex x_i to a factor vertex f_j if and only if x_i is an argument of f_j .

Indicator functions form an important class of functions in coding theory applications of factor graphs. If $P(x_1, \dots, x_n)$ is some predicate involving variables x_1, \dots, x_n , the indicator

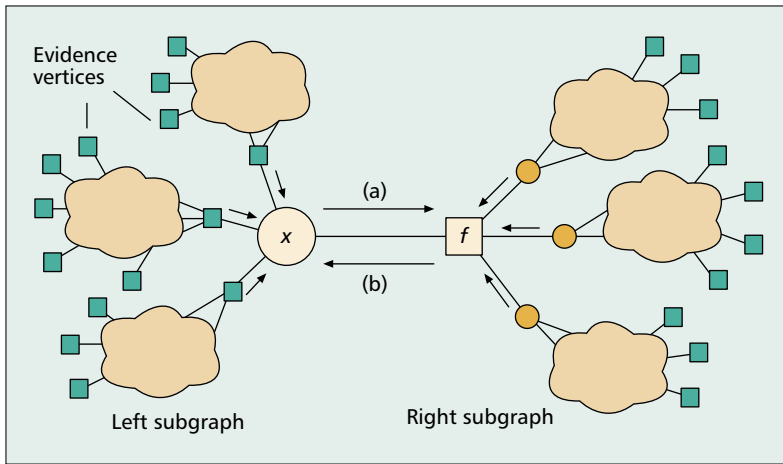
function for P , denoted $\delta[P]$, takes value 1 on configurations of x_1, \dots, x_n for which P evaluates to true, and 0 otherwise. For example, $\delta[x_1 \oplus x_2 \oplus \dots \oplus x_n = 0]$ evaluates to 1 if and only if the binary variables x_1, \dots, x_n form a configuration having even parity (an even number of ones).

Since the product of a number of indicator functions evaluates to 1 only if *all* functions evaluate to 1, such a product represents the logical conjunction (AND) of the corresponding predicates. Thus, if $g(x_1, \dots, x_7)$ is the product of $f_1 = \delta[x_1 \oplus x_2 \oplus x_4 \oplus x_5 = 0]$, $f_2 = \delta[x_2 \oplus x_3 \oplus x_4 \oplus x_6 = 0]$, and $f_3 = \delta[x_4 \oplus x_5 \oplus x_6 \oplus x_7 = 0]$, g evaluates to 1 if and only if f_1 and f_2 and f_3 *all* evaluate to 1; that is, if and only if the variables x_1, \dots, x_7 satisfy the three parity-check equations that define the (7, 4) Hamming code. Thus, g can be viewed as a code-membership indicator function, $g = \delta[(x_1, \dots, x_7) \in C]$, where C denotes the (7, 4) Hamming code. It follows that the graph of Fig. 1a can be understood in two ways: as a Tanner graph for the Hamming code, or as a factor graph for the Hamming code-membership indicator function.

Up to scale, a code-membership indicator function may also be viewed as a probability mass function, representing a distribution over binary n -tuples that is uniform over codewords, and zero on non-codewords. Suppose that a codeword $\mathbf{x} = (x_1, \dots, x_n) \in C$ is chosen according to such a distribution, and transmitted over a memoryless channel, resulting in a received vector $\mathbf{y} = (y_1, \dots, y_n)$. The memoryless property implies that the joint probability mass (or density) function $p(\mathbf{y}, \mathbf{x})$ factors as $p(\mathbf{y}, \mathbf{x}) = p(\mathbf{x}) \prod_{i=1}^n p(y_i | x_i)$, where $p(\mathbf{x})$ is a scaled version of $\delta[\mathbf{x} \in C]$. For any given received \mathbf{y} , the *a posteriori* probability mass function $p(\mathbf{x} | \mathbf{y})$ is given, up to scale, by the function $g(x_1, \dots, x_n) = \delta[(x_1, \dots, x_n) \in C] \prod_{i=1}^n p(y_i | x_i)$, where x_1, \dots, x_n are arguments of g , and the given y_1, \dots, y_n are taken as fixed parameters (for this given decoding problem).

Of course, $g(x_1, \dots, x_n)$ has a nice factorization structure, so it can be represented by a factor graph. Such a factor graph would have

Although we have seen that factor graphs can be used to model codes, these graphs derive their name from the fact that they can, in general, represent the factorization structure of a function of many variables.



■ **Figure 5.** The sum-product algorithm operating in a tree: a) the left-to-right message $m_{x \rightarrow f}$ summarizes evidence gathered in the left subgraph; b) the right-to-left message $m_{f \rightarrow x}$ summarizes evidence gathered in the right subgraph.

precisely the structure of a factor graph for C (i.e., for the function $\delta[(x_1, \dots, x_n) \in C]$), except that attached to the variable vertex x_i is a pendant factor vertex representing $p(y_i|x_i)$. Such *dongle* vertices represent the evidence that must be processed in any given decoding problem. Such evidence vertices are shown schematically in Fig. 5.

THE SUM-PRODUCT ALGORITHM IN CYCLE-FREE FACTOR GRAPHS

Suppose now that we are interested in computing the *a posteriori* probability mass function $p(x_i|\mathbf{y})$ for symbol x_i . Symbol-by-symbol maximum *a posteriori* (MAP) decoding requires such a computation so that the most likely value for x_i may be selected. Clearly, since the function $g(x_1, \dots, x_n)$ represents, up to scale, the *joint a posteriori* probability mass function, computation of $p(x_i|\mathbf{y})$ requires a *marginalization* operation, that is, computation of

$$g_i(x_i) = \sum_{x_1} \cdots \sum_{x_{i-1}} \sum_{x_{i+1}} \cdots \sum_{x_n} g(x_1, \dots, x_n).$$

Most often, particularly in decoding, we will be interested in computing $g_i(x_i)$ for every $i \in \{1, \dots, n\}$. There is obviously a great deal of commonality between many of these sums, with many terms in common.

The sum-product algorithm is a procedure that can be used to organize the simultaneous computation of marginals, resulting in an efficient and exact computation whenever the underlying factor graph is cycle-free. The sum-product algorithm may be understood as operating by passing messages over the edges of the factor graph. We will now explain the nature of these messages, and precisely how they are computed. A more detailed description of the sum-product algorithm may be found in [3].

We first observe that messages are functions. The edges of a factor graph always connect a variable vertex x to a function vertex f , as illustrated in Fig. 5. Messages can pass in either

direction ($x \rightarrow f$ or $f \rightarrow x$) over this edge. Regardless of the message direction, the message passed over an edge incident on a variable x is always a function over the alphabet on which x is defined (i.e., a function of x). For example, suppose that x is a binary variable defined on the alphabet $\{0, 1\}$. Then messages passed on any edges incident on x will be functions of the form $\mu(x)$; such functions can be specified by the vector $[\mu(0), \mu(1)]$, or, if scale is not important, by the ratio $\mu(0)/\mu(1)$ or $\log[\mu(0)/\mu(1)]$. Thus, a message is a function often specified by giving a list of its values. We will denote the message sent from a variable vertex x to a factor vertex f as $\mu_{x \rightarrow f}(x)$, and the message sent from a factor vertex f to a variable vertex x as $\mu_{f \rightarrow x}(x)$.

Since messages are functions, they can be multiplied (as functions). Thus, if $\mu_1(x)$ and $\mu_2(x)$ are messages, their product $\mu_1(x)\mu_2(x)$ is a well defined function of x . Likewise, if $\mu_1(x)$ and $\mu_2(y)$ are functions of x and y , respectively, their product $\mu_1(x)\mu_2(y)$ is a well-defined function of the pair (x, y) .

The sum-product algorithm is defined in terms of two very simple update rules and one very simple termination rule. We will let $N(v)$ denote the set of neighbors of a vertex v in a factor graph. If $w \in N(v)$, v and w share an edge, and the set $N(v) \setminus w$ denotes the neighbors of v other than w . The update and termination rules are as follows.

Update Rule for Variable Vertices — The message $\mu_{x \rightarrow f}$ sent by a variable vertex x to a neighboring factor vertex $f \in N(x)$ is given by

$$\mu_{x \rightarrow f}(x) = \prod_{h \in N(x) \setminus f} \mu_{h \rightarrow x}(x). \quad (1)$$

In words, the message sent from variable vertex x to neighboring factor vertex f is obtained by multiplying the messages received at x from its neighbors *other than* f .

Update Rule for Factor Vertices — The message $\mu_{f \rightarrow x}$ sent by a factor vertex f to a neighboring variable vertex $x \in N(f) = \{x, y_1, y_2, \dots, y_m\}$ is given by

$$\mu_{f \rightarrow x}(x) = \sum_{y_1} \sum_{y_2} \cdots \sum_{y_m} \left(f(x, y_1, \dots, y_m) \prod_{i=1}^m \mu_{y_i \rightarrow x}(y_i) \right). \quad (2)$$

In words, the message sent from factor vertex f to neighboring variable vertex x is obtained by multiplying f by the messages received at f from its neighbors *other than* x , and then marginalizing the resulting function for x .

Termination Rule — The marginal function for x is obtained as

$$\mu(x) = \prod_{f \in N(x)} \mu_{f \rightarrow x}(x). \quad (3)$$

In words, the marginal function for x is obtained by multiplying together all of the messages received at x .

In a cycle-free factor graph, an outgoing

message at any vertex v may be computed and sent on an edge e as soon as all of the information needed to compute that message — messages that arrive at v on edges other than e — is available at v . It follows that message-passing is initiated at the leaf vertices, since such vertices have all the required information at the very start.

The algorithm terminates once every variable vertex has received a message from each of its neighbors. In a finite cycle-free factor graph with E edges, the termination condition is achieved in no more than $2E$ steps (i.e., it is never necessary to send more than two message over an edge, one in each direction). The following theorem is proved in [3].

Theorem 1 — *In a finite cycle-free factor graph representing a function $g(x_1, \dots, x_n)$, the function $\mu(x_i)$ computed by the sum-product algorithm according to Eq. 3 is the marginal function for x_i .*

When $g(x_1, \dots, x_n)$ represents the conditional probability mass function $p(x_1, \dots, x_n | \mathbf{e})$ for n random variables given some observed evidence \mathbf{e} , the messages passed during the operation of the sum-product algorithm in a cycle-free factor graph have a specific interpretation. As shown in Fig. 5, every edge in a cycle-free factor graph induces a partition of the graph into two subgraphs, a *left subgraph* and a *right subgraph*, and also partitions the evidence into two portions: the *left evidence* \mathbf{e}_l and the *right evidence* \mathbf{e}_r , connected to the left and right subgraphs, respectively. The message $\mu_{x \rightarrow y}(x)$ is equal, up to scale, to $p(x | \mathbf{e}_l)$, the conditional probability mass function for x given the left evidence. Likewise, the message $\mu_{y \rightarrow x}(x)$ is equal, up to scale, to $p(x | \mathbf{e}_r)$, the conditional probability mass function for x given the right evidence. At termination, the product of these messages gives the conditional probability mass function for x given *all* of the evidence.

An important example of a cycle-free factor graph is a trellis. The operation of the sum-product algorithm in the chain-like factor graph that describes a trellis (e.g., that shown in Fig. 3b) results in forward and backward propagation of messages along the edges of the factor graph, resulting in the well-known forward/backward or BCJR algorithm [9]. Forward-propagating messages are often denoted using the symbol α and backward-propagating messages using the symbol β . In view of our general interpretation of messages in a cycle-free factor graph, we see that the α messages represent conditional probability mass functions over the state variables given evidence in the past, and the β messages represent conditional probability mass functions over the state variables given evidence in the future. Thus, the well-known BCJR algorithm is obtained as a special case of the sum-product algorithm, and since the underlying factor graph is cycle-free, Theorem 1 assures us that the result of the computation is the true marginal function (i.e., the true *a posteriori* probability mass function for each symbol).

As pointed out in [3], the standard Kalman smoothing algorithm may also be viewed as the forward/backward sum-product algorithm oper-

ating on a chain-like factor graph, under the assumption that the variables are jointly Gaussian distributed. The Kalman filter and predictor are obtained using the forward-only propagation of messages.

The cycle-free assumption not only assures us of the convergence of the sum-product algorithm according to Theorem 1, but this assumption is also the key to the asymptotic analysis and optimization of LDPC codes. The key assumption is that, at extremely long block lengths, the factor graph for such a code has a large cycle-free neighborhood surrounding each vertex. This implies that the messages passed within this neighborhood satisfy the independence assumptions that make the computation performed by the sum-product algorithm exact, at least for a certain number of iterations determined by the size of the cycle-free neighborhood. The independence of the messages permits one to apply a computationally tractable analysis known as density evolution [10], which can be used to determine whether a given LDPC code ensemble contains codes that converge to zero error probability under iterative decoding and for a certain channel quality.

THE SUM-PRODUCT ALGORITHM IN FACTOR GRAPHS WITH CYCLES

The local update rules, Eqs. 1 and 3, do not depend on the factor graph being cycle-free. Thus, even in the presence of cycles, and even though there is no guarantee that the sum-product algorithm provides exact results, one can apply these rules and hope for the best!

The presence of cycles in the graph results in indefinite propagation of messages, resulting in an iterative algorithm with no natural termination. With cycles in the factor graph, the messages in general lose their interpretation as conditional probability mass functions given some subset of the evidence. Even in cases when the algorithm converges to some fixed point and termination messages are computed according to Eq. 3, the resulting functions are in general *not* the true marginal functions.

Despite these problems, application of the sum-product algorithm to decoding codes defined on graphs with cycles and hoping for the best has proven to be a spectacularly successful approach. Even with cycles in their factor graphs, turbo codes and LDPC codes achieve a performance with sum-product decoding that comes to within a fraction of a decibel of the Shannon limit in binary-input additive white Gaussian noise (BIAWGN) channels. Careful optimization of the graph structure associated with ensembles of irregular LDPC codes has led to capacity-achieving performance on the binary erasure channel, and to performance that is practically indistinguishable from the Shannon limit on BIAWGN channels. For example, Table 1 gives the central results from [11], which show that rate-1/2 irregular low-density parity check codes can be designed that achieve a threshold (infinite block length) performance within 0.0045 dB of the Shannon limit, and a practical (block length 10^7) perfor-

The presence of cycles in the graph results in indefinite propagation of messages, resulting in an iterative algorithm with no natural termination. With cycles in the factor graph, the messages in general lose their interpretation as conditional probability mass functions given some subset of the evidence.

d_i	100	200	8000
Threshold (dB from Shannon limit)	0.0247	0.0147	0.00450
Measured performance (dB from Shannon limit)	0.053	0.040	—

Table 1. Performance of rate-1/2 irregular low-density parity check codes [11] on the binary-input AWGN channel. The parameter d_i denotes the maximum variable vertex degree. Threshold denotes the infinite block length performance, and measured performance denotes simulation results for a length 10^7 code at a bit error rate of 10^{-6} .

mance within 0.04 dB of the Shannon limit at a bit error rate of 10^{-6} .

The sum-product algorithm operates in factor graphs with cycles as follows. The algorithm is initialized by passing a (virtual) unit function over every edge of the factor graph. Reception of this virtual unit function permits the update rules, Eqs. 1 and 2, to be applied at any vertex at any step of the algorithm, even when a particular vertex has not received an actual message on some edge.

Message passing may be thought of as occurring in discrete time, according to a message-passing *schedule* that determines which edges are active and in which direction messages are passed during each step of the algorithm. An example is the *flooding* schedule, which makes each edge active in both directions at each time step.

Low-density parity check codes are commonly decoded according to a schedule that activates all edges in the variable-to-check direction during even-numbered time steps and activates all edges in the check-to-variable direction during odd-numbered time steps.

Turbo codes are commonly decoded according to the following schedule. Using the notation of Fig. 4, let C_1 be the constituent code that constrains (\mathbf{x}, \mathbf{p}) and C_2 the constituent code that constrains $(\Pi\mathbf{x}, \mathbf{p})$. In the standard turbo decoding schedule, in one full iteration of message passing, messages are passed from (\mathbf{x}, \mathbf{p}) to C_1 , then from C_1 to \mathbf{x} , then from (\mathbf{x}, \mathbf{q}) to C_2 , and finally from C_2 back to \mathbf{x} . Of course, within the local decoding of C_1 and C_2 , the constituent code's trellis structure can be exploited, and the trellis processed according to a forward/backward message passing schedule.

Termination of the sum-product algorithm in a factor graph with cycles occurs according to some predetermined condition. For example, in decoding LDPC codes, termination might occur when the algorithm results in symbol decisions that form a valid codeword, or after some maximum number of iterations is reached. Turbo decoding is often applied for some fixed number of iterations. In general, the study of schedules and termination conditions in sum-product decoding is a subject of active research.

FURTHER VARIATIONS AND CONNECTIONS

A number of variations of sum-product decoding exist. The max-product algorithm, for example, operates in a factor graph representing the prod-

uct of non-negative local functions exactly like the sum-product algorithm, except that rather than performing local marginalization (with a sum), the maximum among the local configurations is taken. Keeping track of the local configuration that achieves the maximum, the result of the max-product algorithm in a cycle-free factor graph is the configuration of the variables that maximizes the given function. If the function is a likelihood function, the result is the maximum likelihood configuration, so this algorithm implements maximum likelihood (codeword) decoding.

The max-product algorithm is often conveniently implemented in the logarithmic domain. Operating on the negative log-likelihood function, we obtain the min-sum algorithm, which may be regarded as a generalization of the Viterbi algorithm to general graphs. One may also view the min-sum algorithm as an approximate sum-product algorithm.

Messages can be quantized in various ways, leading to a variety of message-passing algorithms. At one extreme, in the decoding of LDPC codes, messages might be quantized to a single bit, leading to so-called bit-flipping decoding algorithms.

As previously mentioned, density evolution is an important tool for the asymptotic analysis of graph-based codes [10] under message-passing decoding, and can potentially be applied to any family of codes that satisfy the property that a sufficiently large local neighborhood around each graph vertex is asymptotically cycle-free. Density evolution can also be combined with optimization methods as a design tool for codes defined on graphs (e.g., [11]).

True density evolution is, however, computationally quite complex, and a variety of approximations have arisen. A very useful approach is based on so-called extrinsic information transfer (EXIT) charts [12]. EXIT charts give insight into the convergence behavior of turbo-like codes, and can be used in the design of coding schemes for a variety of channels.

While we have emphasized modeling and decoding of codes in this article, the factor graph framework potentially allows one to model the source and channel as well [6]. This has the potential to lead to a unified receiver design such as that described in [13], in which the sum-product algorithm is used not only for decoding but for channel estimation as well. It is in principle possible to model a variety of communication system components (e.g., a source model, a multi-antenna transmission model, interference sources) in the factor graph framework and achieve some potentially powerful processing strategies using the sum-product algorithm with iterative processing. For example, the authors of [14] describe a unification of strategies for multiuser detection in a factor graph framework.

CONCLUSION

Codes defined on graphs and decoded using the sum-product algorithm (or one of many possible variations) appear to be a solution to the problem of approaching fundamental lim-

its in communication with practical decoding complexity. The graphical approach to modeling codes gives useful insight into the operation of a variety of potential decoding algorithms. Furthermore, since factor graphs may be used to model components of a communication system (e.g., the source and the channel) beyond the error control code, there is much potential to develop a unified approach in the factor graph framework to the many signal processing tasks that present themselves in a communication receiver.

ACKNOWLEDGMENTS

The author wishes to thank the anonymous reviewers for their many insightful comments.

REFERENCES

- [1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-correcting Coding and Decoding: Turbo Codes," *Proc. 1993 IEEE ICC*, Geneva, Switzerland, May 1993, pp. 1064–70.
- [2] *IEEE Trans. Info. Theory*, vol. 47, no. 2, Special Issue on Codes on Graphs and Iterative Algorithms, Feb. 2001.
- [3] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. Info. Theory*, vol. 47, Feb. 2001, pp. 498–519.
- [4] R. M. Tanner, "A Recursive Approach to Low Complexity Codes," *IEEE Trans. Info. Theory*, vol. IT-27, Sept. 1981, pp. 533–47.
- [5] N. Wiberg, H.-A. Loeliger, and R. Koetter, "Codes and Iterative Decoding on General Graphs," *Euro. Trans. Telecommun.*, vol. 6, Sept./Oct. 1995, pp. 513–25.

- [6] N. Wiberg, "Codes and Decoding on General Graphs," Ph.D. dissertation, Linköping Univ., Dept. of Elec. Eng., 1996.
- [7] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, "Turbo Decoding as an Instance of Pearl's 'Belief Propagation' Algorithm," *IEEE JSAC*, vol. 16, Feb. 1998, pp. 140–52.
- [8] F. R. Kschischang and B. J. Frey, "Iterative Decoding of Compound Codes by Probability Propagation in Graphical Models," *IEEE JSAC*, vol. 16, Feb. 1998, pp. 219–30.
- [9] L. R. Bahl *et al.*, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Info. Theory*, vol. IT-20, Mar. 1974, pp. 284–87.
- [10] T. J. Richardson and R. L. Urbanke, "The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding," *IEEE Trans. Info. Theory*, vol. 47, Feb. 2001, pp. 599–618.
- [11] S.-Y. Chung *et al.*, "On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit," *IEEE Commun. Lett.*, vol. 5, Feb. 2001, pp. 58–60.
- [12] S. ten Brink, "Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes," *IEEE Trans. Commun.*, vol. 49, Oct. 2001, pp. 1727–37.
- [13] A. P. Worthen and W. E. Stark, "Unified Design of Iterative Receivers Using Factor Graphs," *IEEE Trans. Info. Theory*, vol. 47, Feb. 2001, pp. 843–49.
- [14] J. Boutros and G. Caire, "Iterative Multiuser Joint Decoding: Unified Framework and Asymptotic Analysis," *IEEE Trans. Info. Theory*, vol. 48, July 2002, pp. 1772–93.

BIOGRAPHY

FRANK R. KSCHISCHANG (frank@comm.utoronto.ca) is a professor and Canada Research Chair in the Department of Electrical and Computer Engineering at the University of Toronto, where he has been a faculty member since 1991. His research interests are focused on the area of coding techniques, primarily on algorithms for soft-decision decoding.

Codes defined on graphs and decoded using the sum-product algorithm appear to be a solution to the problem of approaching fundamental limits in communication with practical decoding complexity.