To appear in *European Transactions on Telecommunications and Related Technologies*, special issue on Turbo Coding

Codes and Iterative Decoding on General Graphs

June 28, 1995

Niclas Wiberg (nicwi@isy.liu.se), Hans-Andrea Loeliger, and Ralf Kötter
Department of Electrical Engineering, Linköping University
S-581 83 Linköping, Sweden

Abstract — A general framework, based on ideas of Tanner, for the description of codes and iterative decoding ("turbo coding") is developed. Just like trellis-based code descriptions are naturally matched to Viterbi decoding, code descriptions based on Tanner graphs (which may be viewed as generalized trellises) are naturally matched to iterative decoding. Two basic iterative decoding algorithms (which are versions of the algorithms of Berrou et al. and of Hagenauer, respectively) are shown to be natural generalizations of the forward-backward algorithm (Bahl et al.) and the Viterbi algorithm, respectively, to arbitrary Tanner graphs. The careful derivation of these algorithms clarifies, in particular, which a priori probabilities are admissible and how they are properly dealt with. For cycle codes (a class of binary linear block codes), a complete characterization is given of the error patterns that are corrected by the generalized Viterbi algorithm after infinitely many iterations.

1. Introduction

Until recently, most known decoding procedures for error-correcting codes were based on either algebraically *calculating* the error pattern or on some sort of *tree or trellis search.* With the advent of turbo coding [1], however, a third decoding principle has finally had its breakthrough: *iterative decoding.*

Iterative decoding is not a new idea, though. Most of the key ideas were already present in Gallager's pioneering work on low-density parity-check codes [2]. At its time, Gallager's work was actually considered to indicate a promising direction for both research and applications, but the appearance of sequential decoding on the one hand and the rapid advances in algebraic coding theory on the other hand had soon made it fall into oblivion.[1]

Another pioneering work is Tanner's "recursive approach to low complexity codes" [4], which is a general framework, based on bipartite graphs, for the description and decoding of low-complexity codes such as product codes, low-density parity-check codes, and many others. The main thesis of the present paper is that, with respect to iterative decoding, the natural way of describing a code is by means of a Tanner graph, just like trellis-based code descriptions are natural for Viterbi decoding [5]. In fact, we will see that Tanner graphs (which we will define slightly more general than in [4]) can be viewed as generalized trellises. (More precisely, it is the "time axis" of a trellis that is generalized to a Tanner graph.)

1. The perhaps most significant later work on low-density parity-check codes is [3], where the codes were studied from a complexity-theory viewpoint.

Having introduced Tanner graphs as generalized trellises, it will not come as a surprise that iterative decoding on such graphs can be formulated as a generalized version of trellis decoding. We will describe two such algorithms in detail: the min-sum algorithm and the sum-product algorithm. The former may be seen as a generalization of the Viterbi algorithm and the latter as a generalization of the "forward-backward" algorithm of Bahl et al. [6].

When applied to the turbo codes of Berrou et al. [1] (which are included in our general framework), the sum-product algorithm leads to (a version of) the decoding algorithm of [1], while the min-sum algorithm leads to (a version of) an algorithm that has been proposed in [7] as an easily implemented *approximation* of the algorithm of [1]. It should be noted that both algorithms were outlined by Tanner (and, to some extent, already by Gallager), though sketchily and without reference to trellis decoding.

Trellises yield cycle-free Tanner graphs. Code "realizations" of lower complexity (such as turbo codes) are obtained from Tanner graphs with cycles. We will outline how known bounds on the trellis complexity can be generalized to arbitrary Tanner graphs.

When applied to Tanner graphs with cycles, the operation of the two mentioned algorithms is not well understood. For the class of cycles codes [8, pp. 136-138], however, we will give a complete characterization of all error patterns that are corrected by the generalized Viterbi algorithm after infinitely many iterations.

We wish to emphasize that the (few) examples of this paper are not chosen for their performance, but only to illustrate concepts.

The paper is structured as follows. In Section 2, we introduce Tanner graphs. The two basic iterative algorithms are described in Section 3, which contains also the analysis of the min-sum algorithm for cycle codes. Section 4, finally, contains some conclusions. Appendix A outlines the mentioned generalization of bounds on trellis complexity to arbitrary Tanner graphs. The proofs are collected in Appendix B. Some simulation results are given in Appendix C.

2. Tanner Graphs

In this section we propose a framework for code descriptions, or "realizations", that are suitable for iterative decoding. The basic ideas of this approach are due to Tanner [4]. (Note, however, that our terminology and notation differ from that of [4].) We extend Tanner's work by making explicit the connections to trellis coding on the one hand and to system theory on the other hand. We begin with two informal examples.

Figures 1 (right) and 2 (bottom) show two different Tanner graphs for the same binary linear (7, 3, 4) code. A parity check matrix for this code is shown in Figure 1 (left). Note that the last three rows of this matrix are linearly dependent from the other rows and thus redundant.

The (Tanner) graph in Figure 1 (right) is simply an illustration of the structure of that parity check matrix. The codeword components are represented by circular nodes—which we will refer to as *sites*—and each row of the matrix is represented by a small dot which is connected to those sites (i.e., codeword components) that are checked by that row.

$$H = \left[\begin{array}{c} 1\,0\,1\,0\,0\,0\,1 \\ 1\,1\,0\,1\,0\,0\,0 \\ 0\,1\,1\,0\,1\,0\,0 \\ 0\,0\,1\,1\,0\,1\,0 \\ \hline 0\,0\,0\,1\,1\,0\,1 \\ 1\,0\,0\,0\,1\,1\,0 \\ 0\,1\,0\,0\,0\,1\,1 \end{array}\right]$$

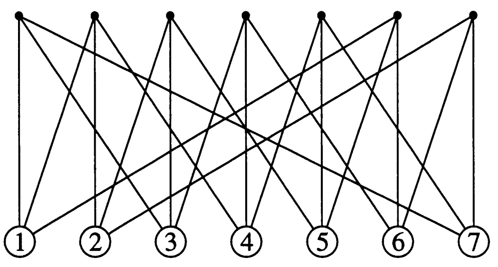


Figure 1 *A parity-check matrix realization of the binary (7, 3, 4) code. (The numbers correspond to the matrix columns.)*

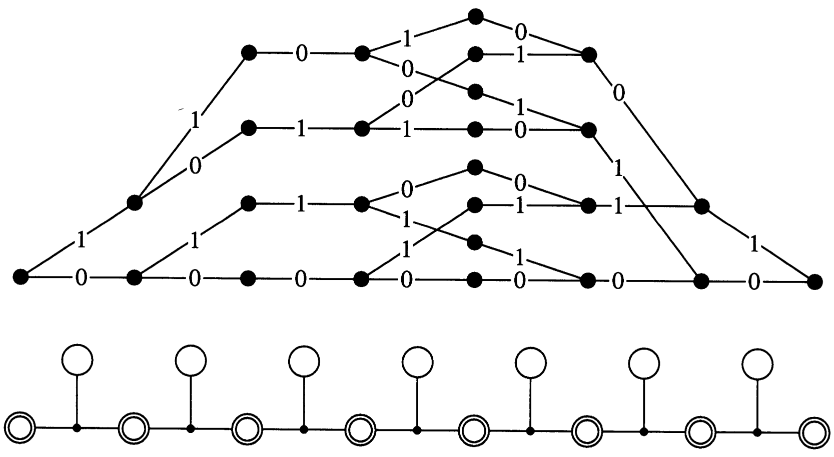


Figure 2 A trellis realization of the binary (7, 3, 4) code.

Figure 2 shows a trellis (top) for the same code together with the Tanner graph (bottom) for that trellis. We now have two types of sites: *visible sites*, which correspond to codeword components, and *hidden sites*, which are depicted as double circles and correspond to the *state spaces* of the trellis. The small dots now stand for the individual sections of the trellis.

We will soon give the formal definition of (our extended version of) a Tanner graph as (the illustration of) a "check structure" for a "system". We hope that the reader will not be repelled by our system-theoretic terminology, the advantages of which will only gradually become obvious.

2.1 Systems and Check Structures

A *configuration space* is a direct product $W = \prod_{s \in N} A_s$, where $\{A_s\}_{s \in N}$ is a collection of *alphabets* (or *state spaces*). In the majority of our examples (e.g., Figure 1), the *index set* N will consist of the integers $\{1, \ldots, n\}$ and $A_s = F_2$ (the binary field) for all $s \in N$, i.e., the configuration space is the familiar space F_2^n of binary n-tuples. More general configuration spaces are, however, essential, e.g., for trellis-based constructions, where some of the alphabets A_s are actually state spaces of the trellis (cf. Figure 2).

We will usually assume that the index set N as well as all alphabets A_s, $s \in N$, are finite. (Neither of these assumptions are essential, though.) Elements of W will be called *configurations*. Elements of N will be referred to as *sites*.

The components of a configuration $x \in W$ will be denoted by x_s, $s \in N$. More generally, the restriction (projection) of $x \in W$ to a subset $R \subseteq N$ of sites will be denoted by x_R. For a set of configurations $X \subseteq W$ and a site subset $R \subseteq N$ we will use the notation $X_R \triangleq \{x_R : x \in X\}$.

A *system* is a triple (N, W, B), where N is a set of sites, W is a configuration space, and $B \subseteq W$ is the *behavior*. (This "behavioral" notion of a system is due to Willems [9], cf. also [10].) The members of B will be called *valid configurations*. A system is *linear* if all alphabets A_s are vector spaces (or scalars) over the same field, the configuration space W is the direct product of the alphabets, and the behavior B is a sub*space* of W.

A *check structure* for a system (N, W, B) is a collection Q of subsets of N (*check sets*) such that any configuration $x \in W$ satisfying $x_E \in B_E$ for all check sets $E \in Q$ is valid (i.e., in B). The restriction (projection) B_E of the behavior to a check set E is called the *local behavior* at E. A configuration x is *locally valid* on E if $x_E \in B_E$. Note that a configuration is valid if and only if it is locally valid on all check sets.

A *Tanner graph* is a visualization of a check structure: the Tanner graph (of a given check structure) is a bipartite graph (as in Figure 1), where each site is represented by a circle and a check set E is represented by a dot (small filled circle) that is connected to those sites (circles) that are members of E. Note that a check structure Q does not by itself define a behavior B; the local behaviors B_E are required too.

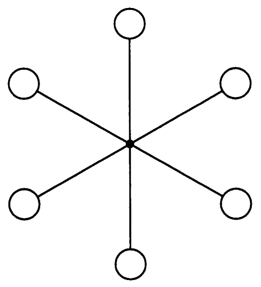


Figure 3 *The trivial check structure (or Tanner graph) with only one check set.*

Any system has a trivial check structure $Q = \{N\}$ consisting of only one check set that contains all the sites, cf. Figure 3. Not surprisingly, this trivial check structure turns out to be unsuitable for iterative decoding. What will be needed are check structures such that the local behaviors B_E, $E \in Q$, are "simple".

Example 1 (parity check matrix), cf. Figure 1. A binary block code C of length n may be viewed as the system (N, W, B), where $N = \{1, 2, \ldots, n\}$, $W = F_2^n$, and $B = C$ is the set of codewords. If H is a parity check matrix for C (i.e., $Hx^T = 0$ if and only if $x \in C$), then the ones in each row of H define a subset of N, and the collection of all these subsets forms a check structure for the system. In this case, all the local behaviors are defined by simple parity checks. A special case of this example are Gallager's low-density parity-check codes [2]. An important special case of Gallager's codes are "cycle codes" [8, pp. 136-138], which are defined by the additional condition that each site belongs to exactly two check sets.

In Example 1, all sites correspond to components of the codewords. However, it is often useful to allow sites that do not correspond to codeword components; such sites will be called *hidden* sites (or *latent* sites, cf. [9]), and the remaining sites will be called *visible*. The system is then denoted by the quadruple (L, V, W, B), where L are the hidden sites and V are the visible sites. A *codeword* of the system is the restriction x_V of a valid configuration $x \in B$ to the visible sites V. The *visible behavior* or *output code* of the system is $B_V \triangleq \{x_V : x \in B\}$. A system (L, V, W, B) with a check structure Q will be called a *realization* of the corresponding output code B_V.

Hidden sites will be depicted by double circles, cf. Figure 2.

Any desired system may be trivially augmented with a single hidden site to allow for a check structure as shown in Figure 4: the value x_s at the single hidden site s takes on a different value for each of the valid configurations $x \in B$. The check structure is defined as $Q = \{\{s, s'\} : s' \in V\}$ where s is the single hidden site, i.e., there is a separate check set for each visible site. It will not surprise the reader that this construction, too, is unsuitable for iterative decoding. What will be needed are hidden sites with "small" alphabets (state spaces).

Example 2 (trellis), cf. Figure 2. A familiar type of hidden sites are the state spaces of a trellis, as is illustrated by Figure 2. The hidden sites correspond to the state spaces and the visible sites correspond to the output symbols. There is a check set for each trellis section, and the corresponding local behavior is simply the set of branches of that trellis section.

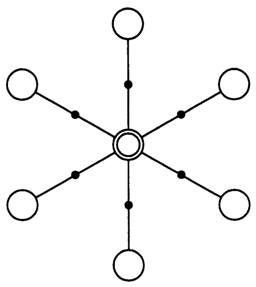


Figure 4 *Another trivial check structure.*

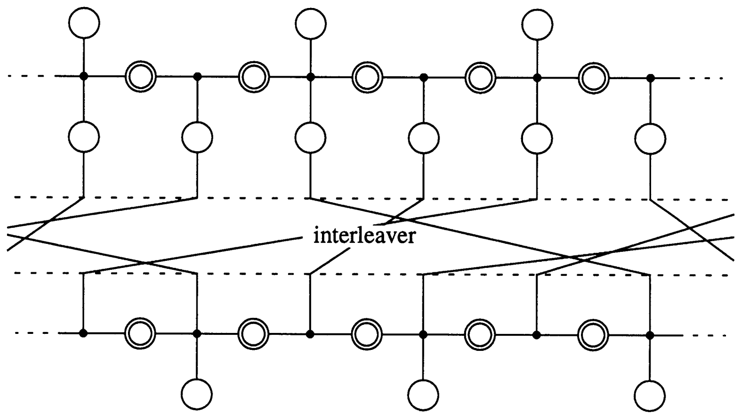


Figure 5 *The Tanner graph of the turbo codes of Berrou et al [1].*

The motivation for introducing hidden sites—and indeed for studying check structures at all—is to find code realizations that are suitable for iterative decoding. At this point, little is known about how the realization affects the decoding performance. However, it seems reasonable that a realization that is to be used for iterative decoding should have

i. small check sets with simple local behaviors

ii. not too many hidden sites, and small hidden state spaces

iii. no short cycles in the Tanner graph.

The first two requirements follow from the fact that the complexity of iterative decoding is roughly proportional to the number and alphabet size of the sites as well as to the number of check sets and to the complexity of the local behaviors. The third requirement follows from the fact that iterative decoding (in the form considered in this paper) is *optimal* for cycle-free graphs (cf. Section 3), and its application to graphs with cycles is (for the time being) primarily justified by good experimental results. Moreover, the minimum distance of the code is often determined by the length of the shortest cycle. (This is well understood for cycle codes [8, pp. 136-138].)

Both the role of hidden sites and the importance of avoiding short cycles are illustrated by the turbo codes of Berrou et al. [1], the Tanner graph of which is shown in Figure 5. The system consist of two trellises that share certain output symbols via an "interleaver" (i.e., the order of the common symbols in one trellis is a permutation of the order in the other trellis). It is well known that the amazing performance of these codes is primarily due to the interleaver, i.e., due to the cycle structure of the Tanner graph.

For binary codes, an interesting special class of realizations are those that have only binary hidden sites (i.e., *all* sites are binary). Such realizations may be considered as punctured low-density parity-check codes. One such example is shown in Figure 6. Its Tanner graph consists of two layers of visible sites and an intermediate layer of hidden sites.

The point of considering Tanner graphs with cycles (as opposed to cycle-free graphs such as trellises) is that the overall complexity of the realization is reduced. What happens, roughly, is that a single trellis state space of size m is split into a number of state spaces of size m_i such that $\prod_i m_i \approx m$. This issue is discussed further in Appendix A, where it is proved (for linear codes) that $\prod_i m_i \geq m$.

In graph theory, the length of the shortest cycle is called *girth*, and a great deal is known on graphs of maximal girth [11][12]. Many possibilities exist for the construction of codes from such extremal graphs. A few such constructions were given in [13] and [14].

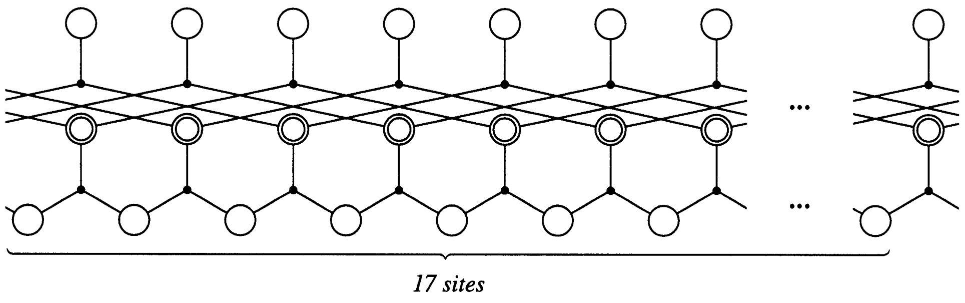


Figure 6 A realization of a (34, 17, 5) quasi-cyclic binary linear code. The Tanner graph "wraps around" at the sides.

3. Two Basic Iterative Algorithms

Let (N, W, B) be a system with a check structure Q. We will describe two basic iterative algorithms, which may be viewed as generalized versions of the Viterbi algorithm and the forward-backward[1] algorithm of Bahl et al. [6], respectively.

The algorithms take as input a set of real-valued *local cost functions*[2] and produce a similar set of *final cost functions* as output. The purpose of the final cost functions is to concentrate all appropriate information from the local cost functions into each site and check set. (For cycle-free check structures, this is done in an "optimal" way, as we will see.)

There is one local cost function for each site $s \in N$, denoted by $\gamma_s : A_s \to \mathcal{R}$ (where $\mathcal{R}$ denotes the real numbers), and one for each check set $E \in Q$, denoted by $\gamma_E : W_E \to \mathcal{R}$. Similarly, there is one final cost function for each site $s \in N$, denoted by $\mu_s : A_s \to \mathcal{R}$, and one for each check set, denoted by $\mu_E : W_E \to \mathcal{R}$. The overall structure of the algorithms is illustrated by Figure 7.

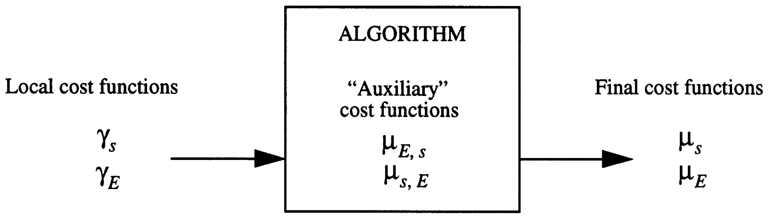


Figure 7 The overall structure of the algorithms.

During the computation, the algorithms maintain a set of "auxiliary" cost functions: for each pair (s, E) of adjacent site and check set (i.e., $s \in E$), there is one "check-set-to-site" cost function $\mu_{E,s} : A_s \to \mathcal{R}$ and one "site-to-check-set" cost function $\mu_{s,E} : A_s \to \mathcal{R}$. As illustrated by Figure 8, the cost $\mu_{E,s}(a)$ may be interpreted as the contribution from E to s on the cost of assigning $x_s = a$. Similarly, the cost $\mu_{s,E}(a)$ may be thought of as the contribution from s to E on the cost of the same assignment. (These interpretations will be made precise for the case of cycle-free check structures.)

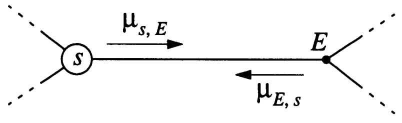


Figure 8 The auxiliary cost functions $\mu_{s,E} : A_s \to \mathcal{R}$ and $\mu_{E,s} : A_s \to \mathcal{R}$.

1. This algorithm is much used in signal processing (hidden Markov models) [15], where it is attributed to L. E. Baum.
2. It may be easier to think of "cost vectors".

3.1 The Min-Sum Algorithm

The min-sum algorithm is a straightforward generalization of the Viterbi algorithm [5]. (The resulting algorithm is essentially Tanner's Algorithm B [4]; Tanner did not, however, observe the connection to Viterbi decoding.) Hagenauer's low-complexity turbo decoder [7] fits directly into this framework. Also a well-known decoding algorithm for generalized concatenated codes [16] is related, as is threshold decoding [17].

The goal of the min-sum algorithm is to find a valid configuration $x \in B$ such that the sum of the local costs (over all sites and check sets) is as small as possible. When using the min-sum algorithm in a channel-decoding situation with a memoryless channel and a received vector y, the check-set costs $\gamma_E(x_E)$ are typically omitted (set to zero) and the local site costs $\gamma_s(x_s)$ are the usual channel log-likelihoods $-\log p(y_s|x_s)$ (for visible sites; for hidden sites they are set to zero). (Alternatively, we could set $\gamma_E(x_E)$ to "infinity" for locally invalid configurations and minimize over the whole configuration space W.) On the binary symmetric channel, for example, the local site cost $\gamma_s(x_s)$ would be the Hamming distance between x_s and the received symbol y_s.

The algorithm consists of the following three steps:

- *Initialization.* The local cost functions γ_s and γ_E are initialized as appropriate (using, e.g., channel information). The auxiliary cost functions $\mu_{E,s}$ and $\mu_{s,E}$ are set to zero.
- *Iteration.* The auxiliary cost functions $\mu_{s,E}$ and $\mu_{E,s}$ are alternatingly updated a suitable number of times as follows (cf. also Figure 9). The site-to-check-set cost $\mu_{s,E}(a)$ is computed as the sum of the site's local cost and all contributions coming into s *except the one from E*:

$$\mu_{s,E}(a) := \gamma_s(a) + \sum_{\substack{E' \in Q: \\ s \in E', E' \neq E}} \mu_{E',s}(a). \tag{1}$$

  The check-set-to-site cost $\mu_{E,s}(a)$ is obtained by examining all locally valid configurations on E that match a on the site s, for each summing the check-set's local cost and all contributions coming into E *except the one from s*. The minimum over these is taken as the cost $\mu_{E,s}(a)$:

$$\mu_{E,s}(a) := \min_{x_E \in B_E : x_s = a} \left[\gamma_E(x_E) + \sum_{s' \in E : s' \neq s} \mu_{s',E}(x_{s'})\right]. \tag{2}$$

- *Termination.* The final cost functions μ_s and μ_E are computed as follows. The final site cost $\mu_s(a)$ is computed as the sum of the site's local cost and all contributions coming into s, i.e.,

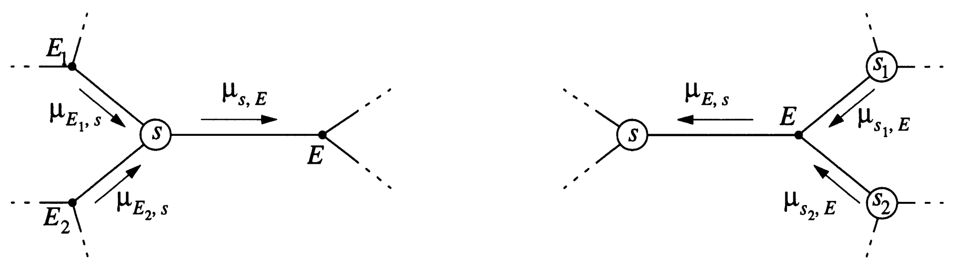
$$\mu_s(a) := \gamma_s(a) + \sum_{E' \in Q : s \in E'} \mu_{E',s}(a), \tag{3}$$

  and the final check-set cost $\mu_E(a)$ is computed as the sum of the check-set's local cost and all contributions coming into E, i.e.,

$$\mu_E(a) := \gamma_E(a) + \sum_{s' \in E} \mu_{s',E}(a_{s'}). \tag{4}$$

Before going into what the min-sum algorithm actually computes, we define the *global cost* for a valid configuration $x \in B$ as

$$G(x) \triangleq \sum_{E \in Q} \gamma_E(x_E) + \sum_{s \in N} \gamma_s(x_s). \tag{5}$$



$$\mu_{s,E}(a) := \gamma_s(a) + \mu_{E_1,s}(a) + \mu_{E_2,s}(a) \qquad \mu_{E,s}(a) := \min_{\substack{x_E \in B_E \\ x_s = a}} [\gamma_E(x_E) + \mu_{s_1,E}(x_{s_1}) + \mu_{s_2,E}(x_{s_2})]$$

***Figure 9** The updating rules for the min-sum algorithm.*

In a typical channel-decoding situation where the check-set costs $\gamma_E(x_E)$ are set to zero and the local site costs are $\gamma_s(x_s) \triangleq -\log p(y_s | x_s)$, the global cost $G(x)$ becomes the log-likelihood $-\log p(y|x)$ for the codeword x; then maximum-likelihood decoding corresponds to finding a valid configuration $x \in B$ that minimizes $G(x)$. As we will see, the min-sum algorithm does this minimization when the check structure is cycle free. In addition, it is also possible to assign nonzero values to the check-set costs $\gamma_E(x_E)$ in order to include an a priori distribution over the codewords: if we define γ_E such that $-\log p(x) = \sum_{E \in Q} \gamma_E(x_E)$, then the global cost will be $G(x) \triangleq -\log p(x) - \log p(y|x) = -\log p(x|y) - \log p(y)$, and minimizing $G(x)$ is equivalent to maximizing the a posteriori codeword probability $p(x|y)$. (See the next section for more about a priori probabilities.)

The following theorem is the fundamental theoretical property of the min-sum algorithm:

Theorem 1 If the check structure is finite and cycle free, then the cost functions converge after finitely many iterations and the final cost functions become

$$\mu_s(a) = \min_{x \in B : x_s = a} G(x) \tag{6}$$

and

$$\mu_E(a) = \min_{x \in B : x_E = a} G(x). \tag{7}$$

The proof is given in Appendix B.1.

Usually, we also want to find the "best" configuration (rather than only its cost). Such a configuration is obtained by taking, for each site s, a value $x_s \in A_s$ that minimizes the final cost $\mu_s(x_s)$. (It may happen that for some sites several site values minimize the final cost for some sites; then it may be non-trivial to find a valid configuration that minimizes μ_s at all sites. For a cycle-free check structure, however, there is a straightforward procedure to solve this problem: start in a leaf site s (one that belongs to a single check set) and choose an optimal value for x_s; then extend the configuration successively to neighbor sites, always choosing site values that both are valid and minimizes the final cost.)

In a practical implementation it is important to handle numerical issues properly. Typically, the cost functions $\mu_{E,s}$ and $\mu_{s,E}$ grow out of range quickly. To overcome this, an arbitrary normalization term may be added to the updating formulas without affecting the finally chosen configuration.

3.2 The Sum-Product Algorithm

The sum-product algorithm is a straightforward generalization of the forward-backward algorithm for the computation of per-symbol a posteriori probabilities in a trellis. Two other special cases of the sum-product algorithm are the classical "turbo" decoder by Berrou et al. [1] and one of Gallager's decoding algorithms for low-density parity-check codes [2]. The general case was outlined by Tanner [4], who did not, however, consider a priori probabilities.

In the sum-product algorithm, the local cost functions γ_s and γ_E have a *multiplicative* interpretation: we define the global "cost" for any configuration $x \in W$ as

$$G(x) \triangleq \prod_{E \in Q} \gamma_E(x_E) \prod_{s \in N} \gamma_s(x_s) . \tag{8}$$

The term "cost" is somewhat misleading in the sum-product algorithm, since it is usually subject to maximization (rather than minimization as in the min-sum algorithm); we have chosen this term to keep the two algorithms as parallel as possible. Also, the algorithm does not maximize G directly; it merely computes certain "local projections" of G, which in turn are natural candidates for maximization.

When discussing the sum-product algorithm, we will usually not consider the behavior B explicitly. Instead we will require that the check-set costs $\gamma_E(x_E)$ are zero for local configurations that are non-valid, i.e., we require

$$\gamma_E(x_E) = 0 \text{ for } x_E \notin B_E . \tag{9}$$

In the typical channel-decoding situation, with a memoryless channel and a received vector y, the local site costs $\gamma_s(x_s)$ are set to the channel likelihoods $p(y_s | x_s)$ (for visible sites; for hidden sites γ_s is set to one), and the local check-set costs are chosen according to the a priori distribution for the transmitted configuration x, which must be of the form $p(x) = \prod_{E \in Q} \gamma_E(x_E)$. This form includes Markov random fields [18], Markov chains, and, in particular, the uniform distribution over any set of valid configurations. (The latter is achieved by taking γ_E as the indicator function for B_E, with an appropriate scaling factor.) With this setup, we get $G(x) = p(x)p(y|x) \propto p(x|y)$, i.e., $G(x)$ is proportional to the a posteriori probability of x. We will see later that, if the check structure is cycle free, the algorithm computes the a posteriori probability for individual *site (symbol) values* $p(x_s|y) = \sum_{x' \in B : x'_s = x_s} G(x')$, which can be used to decode for minimal symbol error probability.

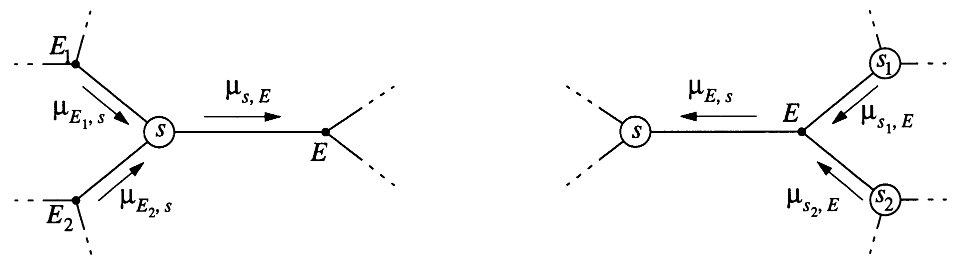
The algorithm consists of the following three steps:

- *Initialization.* The local cost functions γ_s and γ_E are initialized as appropriate (using, e.g., channel information and/or some known a priori distribution). The auxiliary cost functions $\mu_{E,s}$ and $\mu_{s,E}$ are set to one.
- *Iteration.* The auxiliary cost functions $\mu_{E,s}$ and $\mu_{s,E}$ are updated a suitable number of times as follows (cf. also Figure 10). The site-to-check-set cost $\mu_{s,E}(a)$ is computed as the product of the site's local cost and all contributions coming into s *except the one from* E:

$$\mu_{s,E}(a) := \gamma_s(a) \prod_{\substack{E' \in Q: \\ s \in E', E' \neq E}} \mu_{E',s}(a) . \tag{10}$$

The check-set-to-site cost $\mu_{E,s}(a)$ is obtained by summing over all locally valid configurations on E that match a on the site s, each term being the product of the check-set's local cost and all contributions coming into E *except the one from* s:

$$\mu_{E,s}(a) := \sum_{x_E \in W_E : x_s = a} \gamma_E(x_E) \prod_{s' \in E : s' \neq s} \mu_{s',E}(x_{s'}) . \tag{11}$$



$$\mu_{s,E}(a) := \gamma_s(a)\,\mu_{E_1,s}(a)\,\mu_{E_2,s}(a) \qquad \mu_{E,s}(a) := \sum_{\substack{x_E \in W_E \\ x_s = a}} \gamma_E(x_E)\,\mu_{s_1,E}(x_{s_1})\,\mu_{s_2,E}(x_{s_2})$$

***Figure 10** The updating rules for the sum-product algorithm.*

Note that the sum in (11) actually runs only over B_E (the locally valid configurations) since $\gamma_E(x_E)$ is assumed to be zero for $x_E \notin B_E$.

- *Termination.* The final cost functions μ_s and μ_E are computed as follows. The final site cost $\mu_s(a)$ is computed as the product of the site's local cost and all contributions coming into s, i.e.,

$$\mu_s(a) := \gamma_s(a) \prod_{E' \in Q : s \in E'} \mu_{E',s}(a), \tag{12}$$

and the final check-set cost $\mu_E(a)$ is computed as the product of the check-set's local cost and all contributions coming into E, i.e.,

$$\mu_E(a) := \gamma_E(a) \prod_{s' \in E} \mu_{s',E}(a_{s'}). \tag{13}$$

The fundamental theoretical result for the sum-product algorithm is the following:

Theorem 2 If the check structure is finite and cycle free, then the cost functions converge after finitely many iterations and the final cost functions become

$$\mu_s(a) = \sum_{x \in B : x_s = a} G(x) \tag{14}$$

and

$$\mu_E(a) = \sum_{x \in B : x_E = a} G(x). \tag{15}$$

The proof is essentially identical with that of Theorem 1, cf. Appendix B.1.

In particular, we have:

Corollary 3 If the global cost function $G(x)$ is (proportional to) some probability distribution over the configuration space, then the final cost functions are (proportional to) the corresponding marginal distributions for the site values and the local configurations. In particular, if $G(x)$ is proportional to the a posteriori probability $p(x|y)$, then the final cost μ_s is proportional to the per-symbol a posteriori probability $p(x_s|y)$ and the final cost μ_E is proportional to the per-check-set a posteriori probability $p(x_E|y)$.

In a decoding (estimation) application, an estimate for the site value x_s is obtained by choosing the value x_s that maximizes the final cost $\mu_s(x_s)$. With a cycle-free check structure, this minimizes the probability of symbol error.

Again, in a practical implementation, it is important to handle numerical issues properly. Typically, it is necessary to include a normalization factor in the updating formulas in order to prevent the costs from going out of numerical range. This normalization factor does not influence the final maximization.

3.3 Implementation Issues

We have already mentioned that normalization is important to avoid numerical problems. It is also worth mentioning that, in the min-sum algorithm, it is natural to represent costs as *integers* (as is usual with implementations of the Viterbi algorithm). Since the integers are closed under addition and minimization, no precision is lost during the computation.

Another interesting issue is that of updating order. With a cycle-free check structure, each auxiliary cost function need only be updated once: when all its "incoming" cost functions have been computed. Such an updating order is normally used by the forward-backward algorithm and the "soft-output" Viterbi algorithm [19], where the trellis is processed first from left to right and then from right to left. (The original Viterbi algorithm avoids the backwards phase by remembering, for each "state" (i.e., hidden site value), what incoming branch (local configuration) gave the smallest cost; such a scheme can be generalized to any cycle-free check structure.)

Even with a check structure that is not cycle free, certain updating orders may be more natural (or efficient) than others. In the classical "turbo" decoder [1], for example, the two trellises are processed alternatingly, in a forward-backward manner. In Gallager's decoder for low-density codes, on the other hand, the updating is supposed to be done in parallel hardware.

The reader may have noted that, for both the min-sum and the sum-product algorithm, the updating formulas for the site-to-check-set cost functions on the one hand and the check-set-to-site cost functions on the other hand are very similar. In fact, sites could be treated as a special kind of check sets, which would formally unify the update formulas.

3.4 Check Structures that are Not Cycle Free

We are primarily interested in check structures that contain cycles. In this case, the theorems of sections 3.1 and 3.2 do not apply; in fact, the algorithms may not converge at all. It turns out, however, that the final site cost functions usually have a rather clear minimum (or maximum, for the sum-product algorithm). Moreover, the decoding performance of the algorithms is often very good. Indications of this behavior were already present in Gallager's work [2], but the full potential of the algorithms was not obvious (or even conjectured) before the work of Berrou et al. [1].

While a complete analysis of the algorithms is not yet available, it is clear that the theorems may be applied to the first few iterations, i.e., before any cycle has closed. More precisely, for some fixed site s, let $T_s(l) \subseteq N$ consist of those sites (including s itself) that can be reached from s via less than l intermediate sites, where $2l \leq g$ and g is the length of the shortest cycle in Q. If the algorithm is terminated after l iterations, then the algorithm will have operated on a cycle-free "subsystem" $(T_s(l), W_{T_s(l)}, B')$, and thus the theorems of sections 3.1 and 3.2 apply to this subsystem. Note that, in general, the modified behavior B' is different (larger) than the projection of the original behavior to the site set $T_s(l)$ since not all check sets affect the computation.

For one class of codes we can, however, characterize the operation of the min-sum algorithm completely: for the cycle codes [8, pp. 136-138], we can describe exactly what channel outputs give rise to a decoding error. The analysis also suggests that, for these codes, the deviation from true maximum-likelihood decoding is insignificant, which is confirmed by the simulation results in Appendix C.

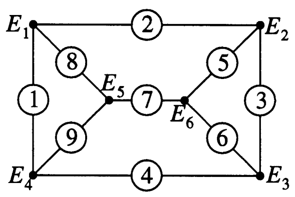


Figure 11 *The Tanner graph of a simple cycle code.*

In our set-up, a cycle code is a binary linear system (N, W, B), $N = \{1...n\}$, $W = F_2^n$, with a check structure Q such that

- every site is contained in exactly two check sets
- all checks are simple parity checks, i.e., $B_E \triangleq \{x_E \in W_E : \sum_{s \in E} x_s = 0\}$, $E \in Q$,

cf. Figure 11. The parameters of these codes are easily determined from the Tanner graph [8, pp. 136-138]. In particular, the minimum distance equals the number of sites in a shortest cycle.

A *non-returning walk*, or simply *walk* on a check structure Q is a sequence $s_1, ..., s_k$ of sites corresponding to a walk on the Tanner graph such that $s_{i+1} \neq s_i$, $i = 1, 2, ...$. A finite walk is *closed* if, when concatenated with itself, it is still a (non-returning) walk. Note that every cycle is a closed walk, but, in general, there are closed walks that are not cycles (e.g., the walk 1,8,7,5,3,6,7,9 in Figure 11). A walk is *irreducible* if it is impossible to obtain a shorter walk by removing a (shorter) closed walk. (Note that the mentioned walk in Figure 11 is irreducible.)

We assume that the local check-set costs γ_E are all zero. Let $\mu_{E,s}^{(l)}$, $\mu_{s,E}^{(l)}$ be the costs in the min-sum algorithm after the l:th iteration.

Theorem 4 The following two conditions are equivalent:

i. $$\sum_{i=1}^{k} \gamma_{s_i}(1) - \gamma_{s_i}(0) > 0 \text{ for all irreducible closed walks } s_1, ..., s_k. \tag{16}$$

ii. $$\mu_{E,s}^{(i)}(1) - \mu_{E,s}^{(i)}(0) \to \infty \text{ for all pairs } (s, E) \text{ (with } s \in E\text{) as } i \to \infty. \tag{17}$$

The proof is given in Appendix B.

Note that if "closed walks" in Condition (i) would be replaced by "cycles", the decoder would be maximum likelihood.

4. Conclusions

The main thesis of this paper is that the framework developed in Section 2 (Tanner graphs) for the description of codes (and "realizations" of codes) is naturally matched to iterative decoding, just like trellises are naturally matched to Viterbi decoding. Within this framework, we have described and explained two basic iterative decoding algorithms, which are generalizations of the forward-backward algorithm and the Viterbi algorithm, respectively. The latter is more amenable to analysis: for cycle codes, we have given a complete characterization of all error patterns that are corrected after infinitely many iterations.

Both algorithms are highly suitable for implementation in parallel hardware. While the min-sum algorithm is better suited to digital hardware (or software), the sum-product algorithm may be suitable for *analog* hardware. Such implementations would look much like "neural networks"; in fact, the prospect of such "neuromorphic" decoders was an important motivation for this work, cf. [20].

Finally, the ubiquity of trellis-based algorithms in signal processing suggests that Tanner graphs and their iterative algorithms will find other applications as well.

Acknowledgment

We are indebted to L. Tolhuizen (from Philips Research Eindhoven, The Netherlands) for pointing out to us reference [4].

Appendix A. State Space Dimensions and Minimality

It is well known that, for a given ordering of the "time" axis, a linear (block or convolutional) code (or, more generally, a group code) has a well-defined unique minimal trellis; any trellis for the same code can be collapsed (by state merging) to the minimal trellis (cf., e.g., [10]). An equivalent statement holds for realizations with an arbitrary cycle-free Tanner graph (which is easily seen by adapting the proofs of [10] to this case).

For Tanner graphs with cycles, however, this is no longer true. E.g., for tail-biting convolutional codes, it is easy to construct examples of inequivalent realizations (in this case: tail-biting trellises) for the same linear code that are minimal in the sense that no states can be merged [20, pp. 34-36]. Nevertheless, the theory of [10] is easily adapted to give a *lower bound* on the size of certain site alphabets (state spaces), which we will now develop.

Let (N, W, B) be a linear system. For disjoint site subsets I and J, the notation $[x_I, y_J]$ will be used for the *concatenation* of x_I and y_J, i.e., $z = [x_I, y_J] \in W_{I \cup J}$ with $z_I = x_I$ and $z_J = y_J$. This notation will also be extended to more than two site sets (which must then be pairwise disjoint). We will also use the notion of a "cut": A *cut* on a check structure Q is a partition $(K\,; I, J)$ of the sites in Q such that no check set in Q contains sites from both I and J. Informally, there is no "walk" from I to J that does not go through K. We will use the following obvious result about cuts:

Lemma 5 Let $(K\,; I, J)$ be a cut on Q. Then any $x \in W$ is valid (i.e., in B) if and only if $x_{K \cup I} \in B_{K \cup I}$ and $x_{K \cup J} \in B_{K \cup J}$.

For a site subset $R \subseteq N$, we define $\tilde{B}_R \triangleq \{x \in B : x_{N \setminus R} = 0\}$, i.e., $\tilde{B}_R$ consists of those valid configurations that are zero outside R. We then have the following definition:

Definition Let (I, J) be a partition of N. The *abstract state space between I and J* is the quotient space $S_{I,J}(B) \triangleq B / (\tilde{B}_I + \tilde{B}_J)$. The *abstract state between I and J* of a valid configuration $x \in B$ is the coset $\sigma_{I,J}(x) \triangleq x + (\tilde{B}_I + \tilde{B}_J)$.

For $N = \{1 \ldots n\}$, the time-j state space of the minimal trellis for B (with the given order of the "time" axis) is in one-to-one correspondence with the abstract state space $S_{\{1 \ldots j\},\, \{j+1 \ldots n\}}(B)$. For general check structures, we only have the following bound:

Theorem 6 Let (N, W, B) be a linear system with the check structure Q. Let $(K\,; I, J)$ be a cut on Q. Then, for any $I' \subseteq I$ and any $J' \subseteq J$, there exists a linear mapping from B_K onto $S_{I',J'}(B_{I' \cup J'})$. In particular, $\dim B_K \geq \dim S_{I',J'}(B_{I' \cup J'})$.

(The proof will be given below.) Of special interest is the case when K contains only hidden sites:

Corollary 7 Let (L, V, W, B) be a linear system with hidden sites and with a check structure Q. Let $(K\,; I, J)$ be a cut on Q such that K contains no visible sites, i.e., $I' \triangleq I \cap V$ and $J' \triangleq J \cap V$ form a partition of V. Then $\sum_{s \in K} \dim A_s \geq \dim S_{I',J'}(B_V)$.

Note that this bound can simultaneously be applied to *all* cuts of the Tanner graph, which results in a linear programming bound on the dimensions of the site state spaces. Note further that bounds on the dimensions of the subcodes $\tilde{B}_R$ may be obtained from the dimension/length profile (generalized Hamming weights) of the code (cf., e.g., [21]), which in turn can be bounded by known bounds on the minimum distance of block codes. The full development of this approach is a challenging research problem.

To prove Theorem 6 we need the following lemma:

Lemma 8 Let (N, W, B) be a linear system and let (I, J) be a partition of N. Then, for arbitrary $x \in B$ and $y \in B$, $[x_I, y_J] \in B$ if and only if $\sigma_{I,J}(x) = \sigma_{I,J}(y)$.

Proof. $[x_I, y_J] \in B \Leftrightarrow [0_I, (y-x)_J] \in B \Leftrightarrow (y-x)_J \in \tilde{B}_J$. Similarly, $[x_I, y_J] \in B \Leftrightarrow (y-x)_I \in \tilde{B}_I$. Together, this gives $[x_I, y_J] \in B \Leftrightarrow y - x \in \tilde{B}_I + \tilde{B}_J$. ❑

Proof of Theorem 6. We first claim that, for any valid configurations x and y such that $x_K = y_K$, we have $\sigma_{I',J'}(x') = \sigma_{I',J'}(y')$, where $x' \triangleq x_{I' \cup J'}$ and $y' \triangleq y_{I' \cup J'}$. Indeed, from Lemma 5, we have $[x_K, x_I, y_J] \in B$ and thus $[x'_{I'}, y'_{J'}] \in B_{I' \cup J'}$, which implies $\sigma_{I',J'}(x') = \sigma_{I',J'}(y')$ by Lemma 8 (applied to $B_{I' \cup J'}$).

The mapping $\varphi : B_K \to S_{I',J'}(B_{I' \cup J'}) : x_K \mapsto \sigma_{I',J'}(x_{I' \cup J'})$ (for arbitrary $x \in B$) is thus well defined. It remains to verify that φ is linear; this follows from $\varphi(\alpha x_K + \beta y_K) = \sigma_{I',J'}((\alpha x_K + \beta y_K)_{I' \cup J'}) = \alpha \sigma_{I',J'}(x_{I' \cup J'}) + \beta \sigma_{I',J'}(y_{I' \cup J'}) = \alpha \varphi(x_K) + \beta \varphi(y_K)$, which holds for any $x, y \in B$ and any scalars α, β. ❑

Appendix B. Proofs

B.1 Proof of Theorems 1 and 2

We will actually only carry out the proof for Theorem 1, i.e., for the min-sum algorithm. The proof is readily adapted to the sum-product algorithm by replacing all sums with products and all minimizations with sums.

Let Q be a cycle-free check structure for the system (N, W, B). For any site subset $R \subseteq N$, we will use the notation

$$G_R(x_R) \triangleq \sum_{E \in Q : E \subseteq R} \gamma_E(x_E) + \sum_{s \in R} \gamma_s(x_s) \tag{18}$$

for the part of the global cost that corresponds to R. We then have $G_N(x) = G(x)$ as a special case.

To prove the theorem, we will start with the claimed expression (6) for the final site cost functions. (The final check-set cost functions will be discussed afterwards.) This expression will be broken down recursively, working "backwards" through the algorithm, until only the local cost functions remain. In order to keep the notation transparent, the proof is carried out only for the specific check structure of Figure 12; the generalization to arbitrary check structures will be obvious.

The final site cost functions μ_s

Consider the final cost $\mu_1(a)$ for some value $a \in A_1$ at site 1 in Figure 12. Let $\nu_1(a)$ denote the "claimed" value for the final cost, i.e., $\nu_1(a) \triangleq \min_{x \in B : x_1 = a} G(x)$. This expression can be written as a sum of three terms, corresponding to the update rule:

$$\nu_1(a) = \min_{x \in B : x_1 = a} G(x) = \min_{x \in B : x_1 = a} \{\gamma_1(a) + G_{R_1}(x_{R_1}) + \gamma_{E_1}(x_{E_1}) + G_{R_2}(x_{R_2}) + \gamma_{E_2}(x_{E_2})\} \tag{19}$$

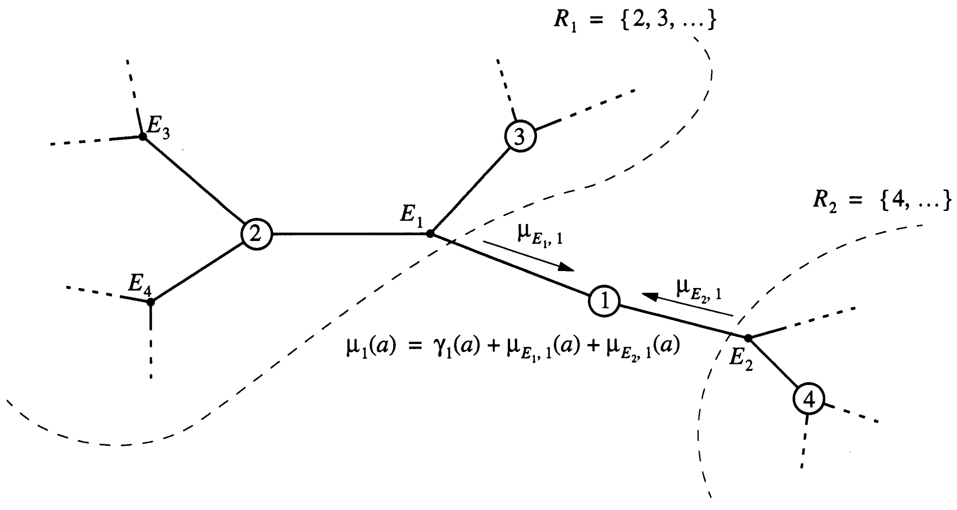


***Figure 12** Computation of the final site cost $\mu_1(a)$.*

$$= \gamma_1(a) + \underbrace{\min_{x_{R_1 \cup \{1\}} \in B_{R_1 \cup \{1\}} : x_1 = a} [G_{R_1}(x_{R_1}) + \gamma_{E_1}(x_{E_1})]}_{\nu_{E_1,1}(a)} + \underbrace{\min_{x_{R_2 \cup \{1\}} \in B_{R_2 \cup \{1\}} : x_1 = a} [G_{R_2}(x_{R_2}) + \gamma_{E_2}(x_{E_2})]}_{\nu_{E_2,1}(a)} \tag{20}$$

This expression has the same structure as the updating rule for the final costs (cf. Figure 12), and we see that the final cost functions indeed get the claimed value ν_1, *provided* that the cost functions coming into site 1 have the right value, i.e., provided that $\mu_{E_1,1} = \nu_{E_1,1}$ and $\mu_{E_2,1} = \nu_{E_2,1}$. Due to the symmetry of the situation, it suffices to consider $\mu_{E_1,1}$.

The check-set-to-site cost functions $\mu_{E,s}$

Figure 13 illustrates the computation of $\mu_{E_1,1}(a)$, which we claim to equal $\nu_{E_1,1}(a)$. The latter can be broken up into three independent parts: the local check-set cost $\gamma_{E_1}(x_{E_1})$ and two costs associated with the site subsets R_2 and R_3, respectively:

$$\nu_{E_1,1}(a) = \min_{x_{R_1 \cup \{1\}} \in B_{R_1 \cup \{1\}} : x_1 = a} [G_{R_1}(x_{R_1}) + \gamma_{E_1}(x_{E_1})] \tag{21}$$

$$= \min_{x_{R_1 \cup \{1\}} \in B_{R_1 \cup \{1\}} : x_1 = a} [\gamma_{E_1}(x_{E_1}) + G_{R_2}(x_{R_2}) + G_{R_3}(x_{R_3})] \tag{22}$$

$$= \min_{x_{E_1} \in B_{E_1} : x_1 = a} \left[\gamma_{E_1}(x_{E_1}) + \underbrace{\min_{x'_{R_2} \in B_{R_2} : x'_2 = x_2} G_{R_2}(x'_{R_2})}_{\nu_{2,E_1}(x_2)} + \underbrace{\min_{x'_{R_3} \in B_{R_3} : x'_3 = x_3} G_{R_3}(x'_{R_3})}_{\nu_{3,E_1}(x_3)} \right]. \tag{23}$$

Again, this expression has the same structure as the updating formula for $\mu_{E_1,1}$, cf. Figure 13. To prove that the updating indeed computes $\nu_{E_1,1}$, we will show that the input costs to the updating rule are the same as the terms of (23), i.e., that $\mu_{2,E_1} = \nu_{2,E_1}$ and $\mu_{3,E_1} = \nu_{3,E_1}$. Due to the symmetry of the situation, it suffices to consider μ_{2,E_1}.

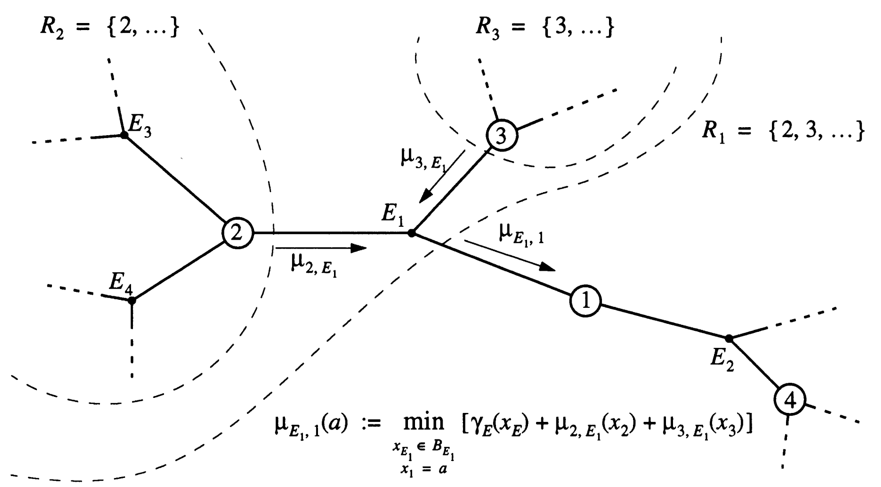


Figure 13 Computation of the check-set-to-site cost $\mu_{E_1,1}(a)$.

The site-to-check-set cost functions $\mu_{s,E}$

Figure 14 illustrates the computation of $\mu_{2,E_1}(a)$, which we claim to equal $\nu_{2,E_1}(a)$. The latter can be broken up into three independent parts: the local site cost $\gamma_2(a)$ and two costs associated with the site subsets R_3 and R_4, respectively:

$$\nu_{2,E_1}(a) = \min_{x_{R_2}\in B_{R_2}:x_2=a} G_{R_2}(a) = \min_{x_{R_2}\in B_{R_2}:x_2=a} \{\gamma_2(a) + G_{R_3}(x_{R_3}) + \gamma_{E_3}(x_{E_3}) + G_{R_4}(x_{R_4}) + \gamma_{E_3}(x_{E_3})\} \tag{24}$$

$$= \gamma_2(a) + \underbrace{\min_{x_{R_3\cup\{2\}}\in B_{R_3\cup\{2\}}:x_2=a} [G_{R_3}(x_{R_3}) + \gamma_{E_3}(x_{E_3})]}_{\nu_{E_3,2}(a)} + \underbrace{\min_{x_{R_4\cup\{2\}}\in B_{R_4\cup\{2\}}:x_1=a} [G_{R_4}(x_{R_4}) + \gamma_{E_3}(x_{E_3})]}_{\nu_{E_4,2}(a)}. \tag{25}$$

Again, this expression has the same form as the updating rule for μ_{2,E_1}, and we only need to show that the input to the updating rule has the value of the last terms of (25). This can be done by going back to the previous step, "*the check-set-to-site cost functions* $\mu_{E,s}$". This process is repeated recursively until the "leaf sites" are reached (those that belong to a single check set), where we observe that the very first updating step computes the desired expression $\nu_{s,E}$:

$$\nu_{s,E}(x_s) = \min_{x_s\in B_s} G_{\{s\}}(x_s) = \gamma_s(x_s). \tag{26}$$

This completes the proof for the final site costs. For the final check-set costs $\mu_E(a)$, only the first step of the proof (the last step of the algorithm) need to be changed, i.e., we need to start in the claimed expression (7) for the final check-set cost. Thus, let E be any check set (in a cycle-free check structure), and let R_s, $s\in E$ be the set of sites that are reachable from E through s. Then the claimed expression can be written as

$$\min_{x\in B:x_E=a} G(x) = \min_{x\in B:x_E=a} \gamma_E(a) + \sum_{s\in E} G_{R_s}(x_{R_s}) = \gamma_E(a) + \sum_{s\in E} \min_{x\in B:x_E=a} G_{R_s}(x_{R_s}), \tag{27}$$

which matches the formula for $\mu_E(a)$, and the recursive process of above can be applied to the terms of the last sum in (27).

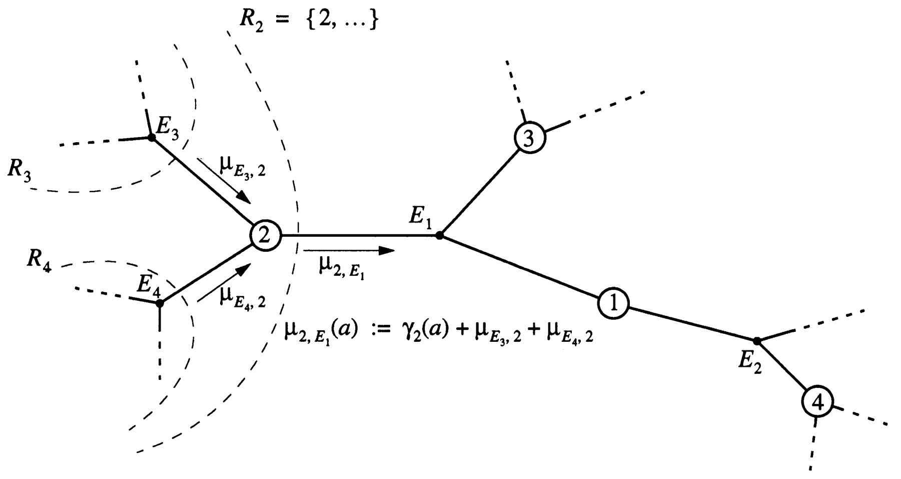


Figure 14 *Computation of the site-to-check-set cost* $\mu_{2, E_1}(a)$.

B.2 Proof of Theorem 4

With every check set E, every site $s \in E$, and every nonnegative integer l, we associate a new system (with a new site set $T_{s, E}(l)$ and a new configuration space) whose Tanner graph is a tree of depth l. As illustrated in Figure 15, this tree is obtained by "unfolding" the Tanner graph of the original system, starting at the site s and the check set E. A site $s' \in N$ gives rise to zero, one, or many copies in $T_{s, E}(l)$. The check structure of the tree system is induced by the check structure Q of the original system; a check set $E' \in Q$ gives rise zero, one, or many tree check sets that inherit the local behavior $B_{E'}$. Every configuration in B gives rise to a valid tree configuration, but (in general), there are valid tree configurations that do not correspond to configurations in B.

For any tree site s', let $\eta(s')$ be the corresponding site of the original system. It is clear from Section 3.1 that

$$\mu_{E, s}^{(l)}(a) = \min_{x'} \sum_{s' \in T_{s, E}(l)} \gamma_{\eta(s')}(x'_{s'}) . \tag{28}$$

where the minimization is over all valid tree configurations x' whose value at the root site equals a.

We now prove (i) $\Rightarrow$ (ii).

Assume that (i) holds. Since the number of irreducible closed walks on Q is finite, (i) implies that there exists a real number $\beta > 0$ such that

$$\sum_{i=1}^{k} \gamma_{s_i}(1) - \gamma_{s_i}(0) > \beta \tag{29}$$

for every irreducible closed walk $s_1, \ldots, s_k$ on Q. There also exists a real number δ such that

$$\sum_{i=1}^{k} \gamma_{s_i}(1) - \gamma_{s_i}(0) > \delta \tag{30}$$

for any irreducible walk $s_1, \ldots, s_k$ (not necessarily closed).

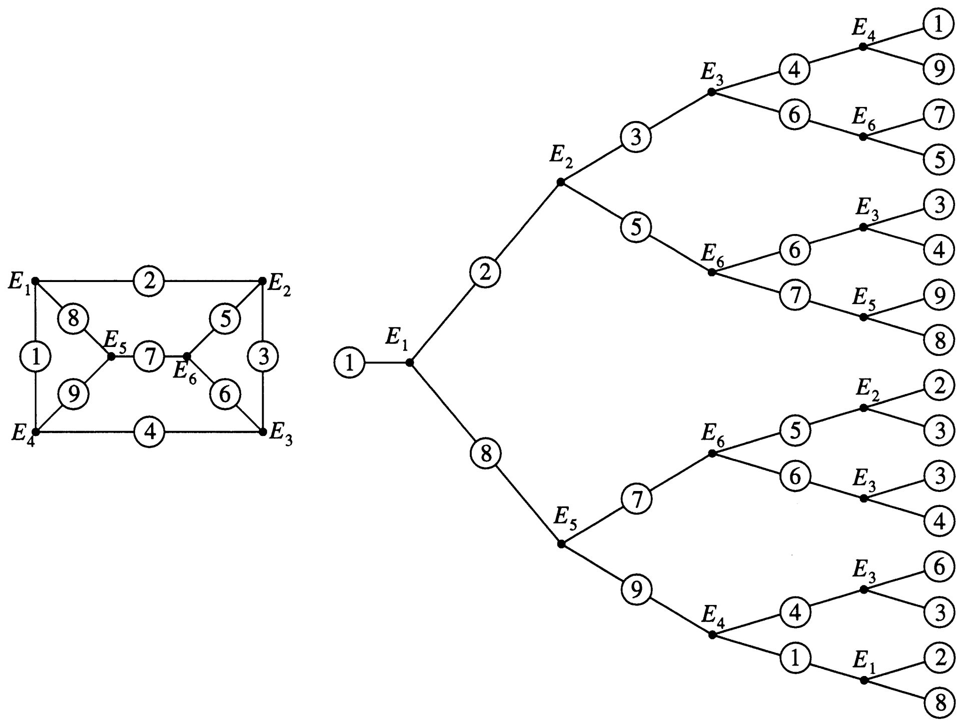


***Figure 15** The tree Tanner graph to the right is obtained from unfolding the Tanner graph to the left, starting at the site $s = 1$, the check set E_1, and $l = 4$.*

Let $x^{(1)}$ be the minimizing x' in (28) for $a = 1$. The definition of a cycle code implies that this tree configuration $x^{(1)}$ contains an all-ones path that starts at the root of the tree and extends to one of the leaves. Inverting all site values along this path yields another valid tree configuration $x^{(0)}$, which is zero at the root site. Let $F \subset T_{s,E}(l)$ be the set of tree sites along this path. Then, from (28),

$$\mu_{E,s}^{(l)}(1) - \mu_{E,s}^{(l)}(0) \geq \sum_{s' \in T_{s,E}(l)} \gamma_{\eta(s')}(x_{s'}^{(1)}) - \gamma_{\eta(s')}(x_{s'}^{(0)}) \tag{31}$$

$$= \sum_{s' \in F} \gamma_{\eta(s')}(x_{s'}^{(1)}) - \gamma_{\eta(s')}(x_{s'}^{(0)}). \tag{32}$$

We next observe that this path corresponds to a non-returning walk on Q. This walk can be "factored" into a number m of irreducible closed walks $F_1, \ldots, F_m$ and a remaining irreducible walk F_0. The length of (i.e., the number of sites in) any component F_i is upper bounded by $2e$, where e is the number of edges of the (original) Tanner graph. Decomposing (32) according to this "factorization" and inserting (29) and (30) yields

$$\mu_{E,s}^{(l)}(1) - \mu_{E,s}^{(l)}(0) \geq m\beta + \delta. \tag{33}$$

But both β and δ are independent of l while $m \to \infty$ for $l \to \infty$, which gives (ii).

It remains to show (ii) $\Rightarrow$ (i).

Assume that (ii) holds. Then, for any site $s \in N$, there exists an integer l_s such that both

$$\mu_{E,s}^{(l)}(0) < \mu_{E,s}^{(l)}(1) \tag{34}$$

and

$$\mu_{s,E}^{(l)}(0) \le \mu_{s,E}^{(l)}(1) \tag{35}$$

for all $l \ge l_s$. Let l_0 be the maximum over these l_s, $s \in N$. For some fixed site $s \in N$ and some $l > l_0$, let x' be the minimizing tree configuration in (28) with $a = 0$. But (34) and (35) imply that x' is all-zero from the root site on to depth $l - l_0$. We record this as a lemma:

Lemma B.1 For any fixed site $s \in N$ and any $l > l_0$, the minimizing tree configuration x' in (28) is zero from the root on to depth $l - l_0$.

We now assume, contrary to the claim of the theorem, that there exists an irreducible closed walk $F_1 \subseteq N$ on Q such that

$$\sum_{s \in F_1} \gamma_{s_i}(1) - \gamma_{s_i}(0) \le 0 ; \tag{36}$$

we will derive a contradiction, which proves the claim. First note that there exists a real number τ such that

$$\sum_{i=1}^{k} \gamma_{s_i}(1) - \gamma_{s_i}(0) < \tau \tag{37}$$

for any non-returning walk $s_1, \ldots, s_k$ on Q of length at most $l_0 + |F_1|$. We now choose a site s in F_1. Let $E \in Q$ be the check set "between" s and its successor in F_1. For some fixed $l > l_0$, let $x^{(0)}$ be the minimizing tree configuration in (28) whose root-site value is zero. We now travel from the root to one of the leaves, along the path F that corresponds to circulating in F_1; along this path, we invert all bits of $x^{(0)}$, which results in a valid tree configuration $x^{(1)}$ with root value one. Applying (28) yields

$$\mu_{E,s}^{(l)}(1) - \mu_{E,s}^{(l)}(0) \le \sum_{s' \in T_{s,E}(l)} \gamma_{\eta(s')}(x_{s'}^{(1)}) - \gamma_{\eta(s')}(x_{s'}^{(0)}) \tag{38}$$

$$= \sum_{s' \in F} \gamma_{\eta(s')}(x_{s'}^{(1)}) - \gamma_{\eta(s')}(x_{s'}^{(0)}) . \tag{39}$$

But Lemma B.1 implies that $x^{(0)}$ is zero from the root on to depth $l - l_0$. Inserting (36) and (37) then yields

$$\mu_{E,s}^{(l)}(1) - \mu_{E,s}^{(l)}(0) \le \tau . \tag{40}$$

Since τ is independent of l, this contradicts our assumption that (ii) holds. ❑

Appendix C. Simulation Results

Simulations were performed for the $(15, 6, 5)$ cycle code of the Peterson graph [8, pp. 136-138] and the $(34, 17, 5)$ code of Figure 6, in both cases comparing the performance of the iterative algorithms to that of maximum-likelihood decoding. As suggested by Theorem 4 (Section 3.4), the performance of iterative decoding is virtually identical to that of maximum-likelihood for the cycle code. For the $(34, 17, 5)$ code, the performance of iterative decoding is slightly inferior to that of maximum-likelihood decoding.

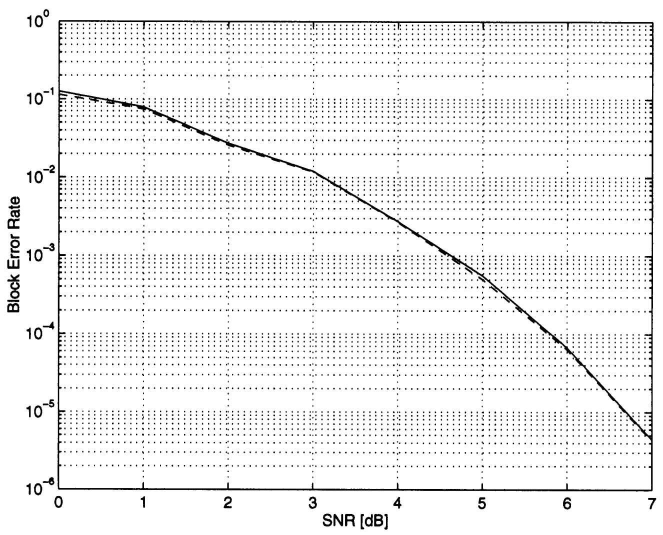


Figure 16 *Decoding performance for the cycle code of the Peterson graph* [8, pp. 136-138]. *The upper curve corresponds to 30 iterations of the min-sum algorithm. (The result for the sum-product algorithm is not shown but is indistinguishable from that of the min-sum algorithm). The bottom curve corresponds to maximum-likelihood decoding.*

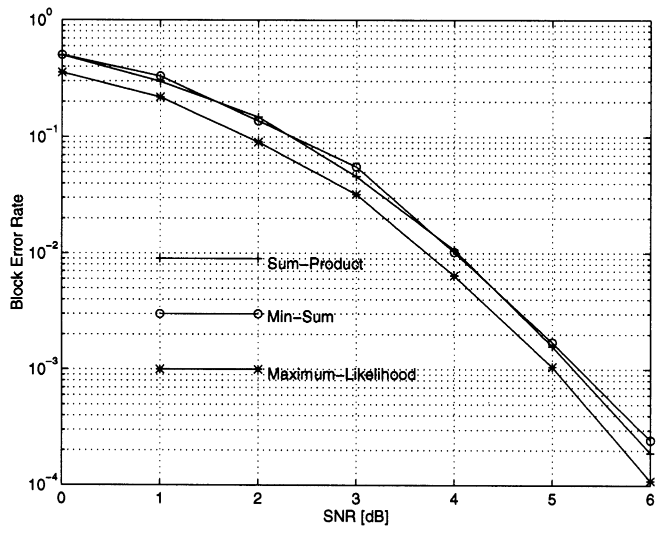


Figure 17 *Decoding performance for the (34,17,5) code of Figure 6. The two upper curves correspond to 30 iterations of the sum-product and the min-sum algorithm, respectively. The bottom curve corresponds to maximum-likelihood decoding.*

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes (1)", *Proc. ICC'93*, Geneva, Switzerland, 1993, pp. 1064-1070.

[2] R. G. Gallager, "Low-density parity-check codes", *IRE Trans. Inform. Theory*, vol. 8, pp. 21-28, Jan. 1962.

[3] V. V. Zyablov and M. S. Pinsker, "Estimation of the error-correction complexity of Gallager low-density codes", *Probl. Peredach. Inform.*, vol. 11, pp. 23-36, Jan. 1975.

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

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

[6] 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. IT-20, pp. 284-287, March 1974.

[7] J. Hagenauer and L. Papke, "Decoding 'turbo'-codes with the soft output Viterbi algorithm (SOVA)", *Proc. 1994 IEEE Int. Symp. Inform. Th.*, Trondheim, Norway, June 27-July 1, 1994, p. 164.

[8] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*. 2nd ed., Cambridge, MA: MIT Press, 1972.

[9] J. C. Willems, "Models for dynamics", in *Dynamics Reported*, vol. 2, U. Kirchgraber and H. O. Walther, eds., Wiley and Teubner, 1989, pp. 171-269.

[10] H.-A. Loeliger, G. D. Forney, T. Mittelholzer, M. D. Trott, "Minimality and observability of group systems", *Linear Algebra & Appl.*, vol. 205-206, pp. 937-963, July 1994.

[11] A. E. Brouwer, A. M. Cohen, A. Neumaier, *Distance-Regular Graphs*, Springer, Berlin, 1989.

[12] P. K. Wong, "Cages—a survey", *J. Graph. Th.*, vol. 6, pp. 1-22, 1982.

[13] J. E. M. Nilsson and R. Kötter, "Iterative decoding of product code constructions", *Proc. ISITA94*, pp. 1059-1064, Sydney, November 1994.

[14] R. Kötter and J. E. M. Nilsson, "Interleaving strategies for product codes", *Proc. EIDMA Winter Meeting on Coding Th., Inform. Th. and Cryptol.*, Veldhoven, Netherlands, Dec. 19-21, 1994, p. 34.

[15] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition", *Proc. IEEE*, vol. 77, pp. 257-286, Feb. 1989.

[16] V. A. Zinoviev and V. V. Zyablov, "Decoding of non-linear generalized concatenated codes", *Probl. Peredach. Inform.*, vol. 14, pp. 46-52, 1978.

[17] J. L. Massey, *Threshold decoding*, MIT Press, Cambridge, MA, 1963.

[18] R. Kindermann and J. L. Snell, *Markov Random Fields and their Applications*. Providence: American Mathematical Society, 1980.

[19] J. Hagenauer, P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications", *Proc. GLOBECOM'89*, Dallas, Texas, pp. 47.1.1-47.1.7, Nov. 1989.

[20] N. Wiberg, *Approaches to Neural-Network Decoding of Error-Correcting Codes*, Linköping Studies in Science and Technology, Thesis No. 425, 1994.

[21] G. D. Forney, Jr., "Density/length profiles and trellis complexity of linear block codes and lattices", *Proc. IEEE Int. Symp. Inform. Th.*, Trondheim, Norway, June 27 - July 1, 1994, p. 339.