Iterative Decoding of Codes Defined on Graphs

by

Alexander Reznik

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1998

Author ....
Department of Electrical Engineering and Computer Science
May 8, 1998

Certified by ....
G. David Forney, Jr.
Adjunct Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Iterative Decoding of Codes Defined on Graphs

by

Alexander Reznik

Submitted to the Department of Electrical Engineering and Computer Science
on May 8, 1998, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

Abstract

Graphs with cycles can provide simpler representations of codes than cycle-free graphs. While iterative decoding is most often used to decode such codes, the properties of this method are not well understood on graphs with cycles.

This thesis first explores an exact approach for decoding on graphs with cycles based on work by Lauritzen and Spiegelhalter [LS88]. It is shown that this approach leads to an exact algorithms with increased complexity.

We then explore the behavior of the min-sum algorithm on a graph with a single loop (a tail-biting trellis). For an AWGN channel we derive a measure of the performance of the algorithm which turns out to be equivalent to the "generalized distance" defined in [Wib96]. We demonstrate the relevance of this "generalized distance" to the performance of the min-sum algorithm on the tail-biting trellis.

Lastly, simulation results for the tail-biting trellis of a (24,12,8) Golay code are presented and discussed in light of the above development. The simulations show that iterative decoding performs approximately as well as a maximum-likelihood decoder.

Thesis Supervisor: G. David Forney, Jr.
Title: Adjunct Professor

Acknowledgments

I would like to thank Prof. Forney without whose support and guidance this project would not have been possible. His advice and encouragement kept me pushing when it seemed that the effort would be fruitless.

I would also like to thank Prof. Trott and Prof. Kschischang for their help; Li Shu for support and hours of valuable discussions; and last, but definitely not least, Inna Levine for love, support and patience.

Contents

Chapter 1

Introduction

1.1 Background

The use of iterative decoding algorithms for codes defined on graphs has become a major direction in research in coding theory over the past few years. These algorithms are key to the incredible performance of "turbo codes" [BGT93], which can achieve error rates of 10^{-5} for SNRs within 0.5 dB of the Shannon limit.

The use of graphical models and iterative decoding algorithms can be traced as far back as Gallager's work on low density parity check (LDPC) codes [Gal62], [Gal63]. Tanner [Tan81] established a framework for defining codes on graphs and developed low-complexity decoding algorithms for such codes.

More recently, Wiberg, Loeliger and Kötter [Wib96], [WLK95] extended the graphical models of Tanner to include hidden nodes. Hidden nodes are variables that cannot be observed directly from the channel (for example, states of a convolutional encoder). This extension allowed the authors to generalize Tanner graphs to include trellis representations of codes. The resulting graphs will be referred to as Tanner-Wiberg-Loeliger (TWL) graphs.

Other recent work has demonstrated connections between TWL graphs and decoding algorithms over these graphs and graphical representations and algorithms that have been developed in other scientific communities. McEliece, MacKay and Chang [MMC98] have shown the connection between the iterative decoding algorithms used for decoding turbo codes and the belief propagation (BP) algorithm used in the artificial intelligence (AI) community for propagating information through the "knowledge tree" of an expert system. Knowledge trees are represented using directed acyclical graphs (DAGs) and BP operates on these graphs (see [Pea88]).

Kschischang and Frey [KF98] have demonstrated the connection between TWL graphs, DAGs and Markov Random Fields (MRFs). MRFs are undirected graphical models used for probability propagation algorithms primarily in the statistics and statistical physics community. The paper unifies these three models under the common framework of "factor graphs."

The different algorithms for the three graphical models (TWL graphs, DAGs and MRFs) are closely related. This is clearly shown in [KF98], but also in [Wib96], [WLK95] and [MMC98]. They all aim to decode subject to local constraints that exist between neighboring nodes in the graph. Information propagates through the graph because certain nodes are "shared", i.e. they belong to a number of groups sharing a local constraint, and thus must be in a configuration such that all of these constraints are satisfied. The algorithms work by using the information observed from the channel, and then disseminating it through the graph in such a way that all the local constraints hold. Since the channel information is usually converted to probabilities (that a particular variable had a particular value at the input to the channel), the algorithm involves propagation of these observed probabilities.

Assuming that the algorithm converges, the end result is a consistent joint probability distribution over all the variables in the graph. This joint probability distribution incorporates in some fashion the observations from the channel. The decoding is completed by picking out either the most probable sequence of variables or the most probable value of each individual variable.

Depending on one's goal, different conditions of optimality can be defined. If one is estimating the original sequence (codeword) that was sent, then an optimal algorithm is the one that results in the maximum likelihood (ML) sequence. If one seeks to get the best estimate of the value of each individual symbol in the sequence, then optimal decoding is the one that results in maximum *a posteriori* (MAP) probability estimation of the symbols. The two algorithms do not in general give the same result. The MAP symbol-by-symbol sequence may be a non-codeword, in which case it could never be the output of a ML sequence detector. When systematic encoding is used (i.e. the original message bits are present in predefined positions in the codeword), symbol-by-symbol MAP estimation results in better performance than ML sequence estimation in terms of bit error probability for the message bits. At high SNR, however, the difference is negligible.

1.2 Types of algorithms

There are two types of iterative algorithms used in decoding, both described in [Wib96], [WLK95].

The min-sum algorithm (also called max-sum in some of the literature) uses log-likelihood channel information to produce an estimate of the transmitted sequence. The bi-directional Viterbi algorithm results if the min-sum algorithm is applied to a trellis, so this algorithm is sometimes referred to as a "generalized Viterbi algorithm."

The sum-product algorithm uses channel information in the form of *a posteriori* probabilities that a symbol has a particular value. It then "propagates" these probabilities with the objective of producing the correct *a posteriori* joint probability distribution of all the variables and thus performing MAP symbol-by-symbol estimation. The sum-product algorithm applied to a trellis results in the BCJR (backward-forward, APP, "MAP") algorithm [BCJR74].

The sum-product algorithm is used in turbo decoding and is closely related to BP [MMC98]. We will see later on in this thesis that it is also related to another probability propagation scheme [LS88], which itself is related to BP [Pea88], [Jor], [KF98].

The min-sum algorithm is sometimes considered to be an approximation to the sum-product. It is usually less computationally complex to implement, and although its performance on a symbol-by-symbol basis is not as good as that of sum-product, it is usually close to that of the sum-product algorithm and asymptotically approaches it. However, the min-sum algorithm is also important in its own right. Much of Chapter 3 of this thesis is devoted to it.

The two forms of the algorithm are intimately related, as is shown by [Wib96], [WLK95], [KF98]. Thus, many "high-level" proofs that apply to one will, with appropriate modifications, apply to the other. For example, this holds for the proof of optimality of these algorithms for cycle-free graphs that is given in [Wib96], [WLK95]. However, when the specifics of these algorithms begin to become essential to proofs, then their relationship becomes more complicated.

1.3 Convergence of iterative algorithms

On graphical models without cycles, the iterative decoding algorithms are well understood. They requires no more then two passes through the complete graph to converge [Wib96], [WLK95], [For97]. Forney [For97] as well as [Wib96] and [WLK95] present a good explanation of the workings of the algorithm. The optimality of the two forms of the algorithm on cycle-free graphs has been shown in [Wib96] and [WLK95], who also prove that only two passes through the graph (one forward and one backward) are necessary for the algorithm to converge and completely update every node.

However, the approach is most powerful on graphical models with cycles, such as apply to turbo codes and LDPC codes. There is little known analytically about how and why iterative algorithms work so well on graphs with cycles.

The basic problem is that on a cycle-free graph, one can start at the leaves and work toward the "root" of the tree. The algorithm then backtracks to the leaves, updating the nodes with the information from the nodes closer to the root; it stops when the leaves are finally updated. By contrast, on a graph containing cycles, one can loop around in the cycles with no guarantee of convergence. Even if the algorithm happens to converge, there is no guarantee that it converges to the ML sequence or MAP symbols.

Recent work has began to address the issues relating to the iterative algorithms for graphs with cycles. For example, [BM96] have been able to address several issues with regards to turbo codes.

A significant amount of work has been done on graphs that contain a single cycle. Anderson and Hladik [AH98] show the convergence of sum-product algorithm on graphs that contain a single cycle. Subsequently, more results were obtained about the sum-product and the min-sum algorithm [Wei97], [FKM98]. Intuition about min-sum algorithm obtained for graphs with a single cycle led to the more general results of [FKV97]. However, even for graphs with a single cycle the picture is still incomplete.

1.4 The results of the thesis

In Chapter 2 of this thesis we will explore a procedure for exact probability propagation through an MRF and apply it to decoding TWL graphs. Throughout this thesis, by *exact* we mean a procedure that is optimal and that converges after a finite number of iterations. This procedure was originally developed by Lauritzen and Spiegelhalter [LS88] for exact probability propagation through DAGs. However, the procedure itself uses MRFs to which the original DAGs are converted.

The Lauritzen and Spiegelhalter (LS) procedure can be described as grouping variables in such a way that a new graph, with these groups of variables as its nodes, is cycle-free, and then using probability propagation on this new graph. We will show that this procedure results in an increase in complexity that makes it useless for most practical applications. In particular, we will show that the complexity of the LS procedure is lowerbounded by the "cut-set bound" of [Wib96]. This means that the LS procedure is no less complex than the sum-product algorithm on a cycle-free TWL graph of the code, on which the sum-product algorithm is also exact.

In Chapter 3 we turn our attention to the case of a graph with a single cycle, called a tail-biting trellis (TBT). We examine the performance of the min-sum algorithm on such a graph and derive an appropriate measure of performance of the min-sum algorithm on an AWGN channel. We generalize our result to more general graphs and show that it is equivalent to the "generalized weight" of [Wib96].

In Chapter 4 we present some simulation results of the min-sum algorithm on a

TBT of a (24,12,8) Golay code [CFV98]. We discuss these simulation results in light of the results of Chapter 3. Our most important observation is that iterative decoding is a near-ML decoding algorithm in this case.

Chapter 2

Relationship between TWL graphs and Markov Random Fields

In this chapter we define TWL graphs and Markov Random Fields (MRFs). The definition of TWL graphs will essentially follow that of [Wib96] and [WLK95], except that we introduce an additional "deterministic" constraint.

We then prove that a probability distribution over an MRF obtained from a TWL graph obeys the Global Markov Property. This result is important since the procedure for exact probability propagation developed in [LS88], which will be presented in the following chapter, requires that the distribution involved obey the Global Markov Property.

Lauritzen [Lau96] shows that a probability distribution obeys the Global Markov Property over an appropriate MRF if it is strictly positive. The distributions involved in coding, however, have many zero values. We prove, using the special structure of these distributions as modeled by TWL graphs, that these distributions obey the Global Markov Property over their MRF representations.

We then proceed to define decomposable graphs and demonstrate how distributions over decomposable graphs can be factorized. This leads us to the procedure for exact probability propagation developed by Lauritzen and Spiegelhalter in [LS88] (to be referred to as the LS procedure). We show that this results in a more computationally complex procedure unless the original TWL graph was already a tree, in which case the LS procedure is equivalent to the sum-product algorithm. Moreover, we show that the complexity of this procedure is lowerbounded by the "cut-set bound" of [Wib96].

2.1 TWL graphs

A TWL graph is an undirected, bipartite graph $\mathcal{T} = (S, C, E_T)$ where S is the set of sites, C is the set of check nodes and E_T is the set of edges. An edge in $\mathcal{T}$ can connect only sites to check nodes, not sites to sites or check nodes to check nodes. A check node and a site are said to be *connected* if there exists an edge between them.

A *site* corresponds to a variable. There are two types of sites, *visible* and *hidden*. A *visible site* corresponds to a variable for which extrinsic information (e.g., a channel observation) is available. A *hidden site* corresponds to a variable for which no extrinsic information is available, e.g., the state of a convolutional encoder. The availability of extrinsic information is the only distinction between visible and hidden sites. This distinction is important from the point of view of the decoding application. However, it is immaterial to the algorithms discussed in this thesis, which treat all variables (sites) equally.

A *check node* represents a constraint on the variables associated with it. For example, for binary codes, a check node may represent a parity check on some symbols of the code.

We restrict TWL graphs to be *deterministic* in the sense that if all the sites connected to a particular check node but one are known, the value of the unknown site is completely determined by the values of the other sites. This property imposes a special structure on the check nodes, which in turn implies a special structure of distributions which can be modeled by TWL graphs.

Two sites α and β are *adjacent* if there exists a check node η such that both α and β are connected to η.

A *path* of length n from a site α to a site β is a set of sites $\{\gamma_0, \gamma_1, ...\gamma_n\}$ such that any pair of sites $\{\gamma_i, \gamma_{i+1}\}, 0 \leq i < n$ are adjacent, and $\alpha = \gamma_0, \beta = \gamma_n$.

A *cycle* of length n is a path of length n where $\gamma_0 = \gamma_n$. Note that a cycle of length 2 implies that there are two sites connected to same two check nodes.

Two disjoint sets of sites, A and B, are *adjacent* if there exists some $\alpha \in A$ and some $\beta \in B$ such that α and β are adjacent.

Two disjoint sets of sites A and B are said to be *separated* by a set of sites S, if every path from every site in A to every site in B contains at least one site in S. If S is disjoint from A and B, S is called a *separator set* for A and B. Note that if A and B are adjacent, then there exists no separator set that is disjoint from both A and B.

2.2 Markov Random Fields

Given an undirected graph $\mathcal{G} = (V, E)$, where V is the set of vertices and E is the set of edges, and a collection of random variables $(X_\alpha, \alpha \in V)$, $\mathcal{G}$ is called a *Markov Random Field (MRF)* if a probability distribution P on $\mathcal{G}$ obeys the *Pairwise Markov Property*, relative to $\mathcal{G}$.

The Pairwise Markov Property is the weakest of three Markov properties which we now define.

A probability distribution P on an undirected graph $\mathcal{G}$ is said to obey:

(P) *the Pairwise Markov Property:* for any pair of non-adjacent vertices (α, β),

$$\alpha \perp \beta | V \setminus \{\alpha, \beta\}$$

i.e. any non-adjacent vertices (α, β) are independent given the rest of the vertices.

(The symbol $\perp$ denotes statistical independence, and the symbol $|$ denotes conditioning.)

(L) *the Local Markov Property:* for any vertex α,

$$\alpha \perp V \setminus cl(\alpha) | bd(\alpha)$$

i.e. α is independent of all the vertices, other than the closure of α, given the boundary of α. Here, the *boundary (bd)* of α is the set of all nodes adjacent to α, and the *closure (cl)* of α is the boundary of α joined with α.

(G) *the Global Markov Property:* for any triple (A, B, S) of disjoint subsets of V such that S separates A and B,

$$A \perp B | S$$

i.e. any node in A is independent of any other node in B, given all the nodes in S.

In the above, and throughout the thesis, we define relationships between random variables of the distribution P by the relationships between the corresponding nodes in the MRF or the TWL graph. For example, $\alpha \perp \beta$ means $X_\alpha \perp X_\beta$ where X_α and X_β are random variables in P.

2.3 Relating TWL graphs and MRFs

As in [KF98], we use the following transition from a TWL graph to an MRF.

i Each site of the TWL graph transforms into a vertex of the MRF. A vertex of an MRF and a site of the TWL graph are said to *correspond* to each other if they correspond to the same variable.

ii An edge is placed between two vertices on the MRF if and only if the corresponding sites of the TWL graph are adjacent.

iii Check nodes of the TWL graph are dropped.

This ensures that any two vertices are adjacent in the resulting MRF if and only if the corresponding sites of the TWL were adjacent. Figures 2.1 and 2.2 illustrate examples of TWL graphs and corresponding MRF representations.

We now have to show that the resulting undirected graph satisfies the Markov properties. Lauritzen [Lau96] shows that **(G)**$\Rightarrow$**(L)**$\Rightarrow$**(P)**. Therefore, we need to show only that the resulting graph satisfies **(G)**.

For notational convenience, we will refer to corresponding vertices (or sets of vertices) of the MRF and sites of the TWL graph (or sets of sites) by the same

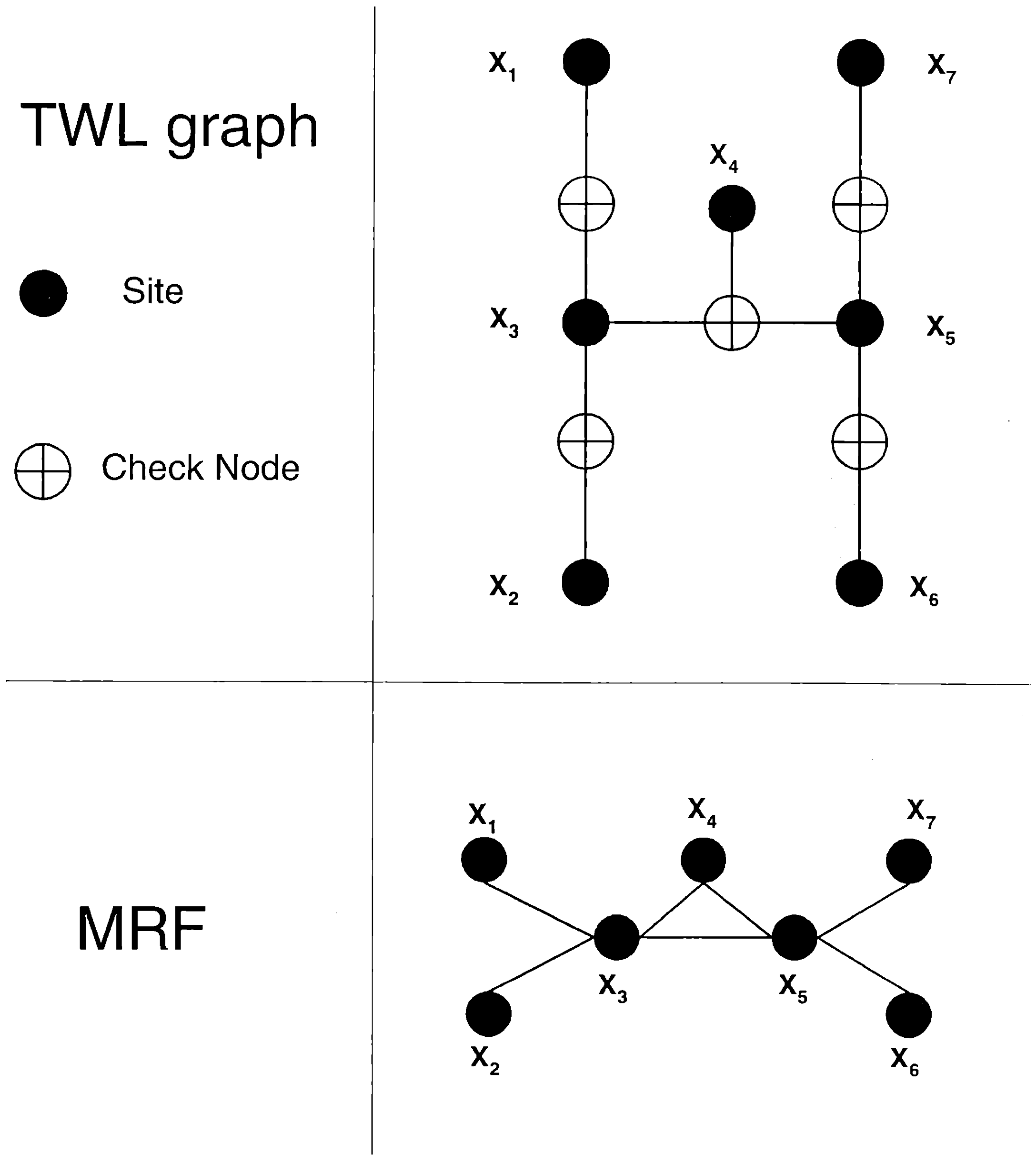

Figure 2.1 Example of a TWL graph and the corresponding MRF.

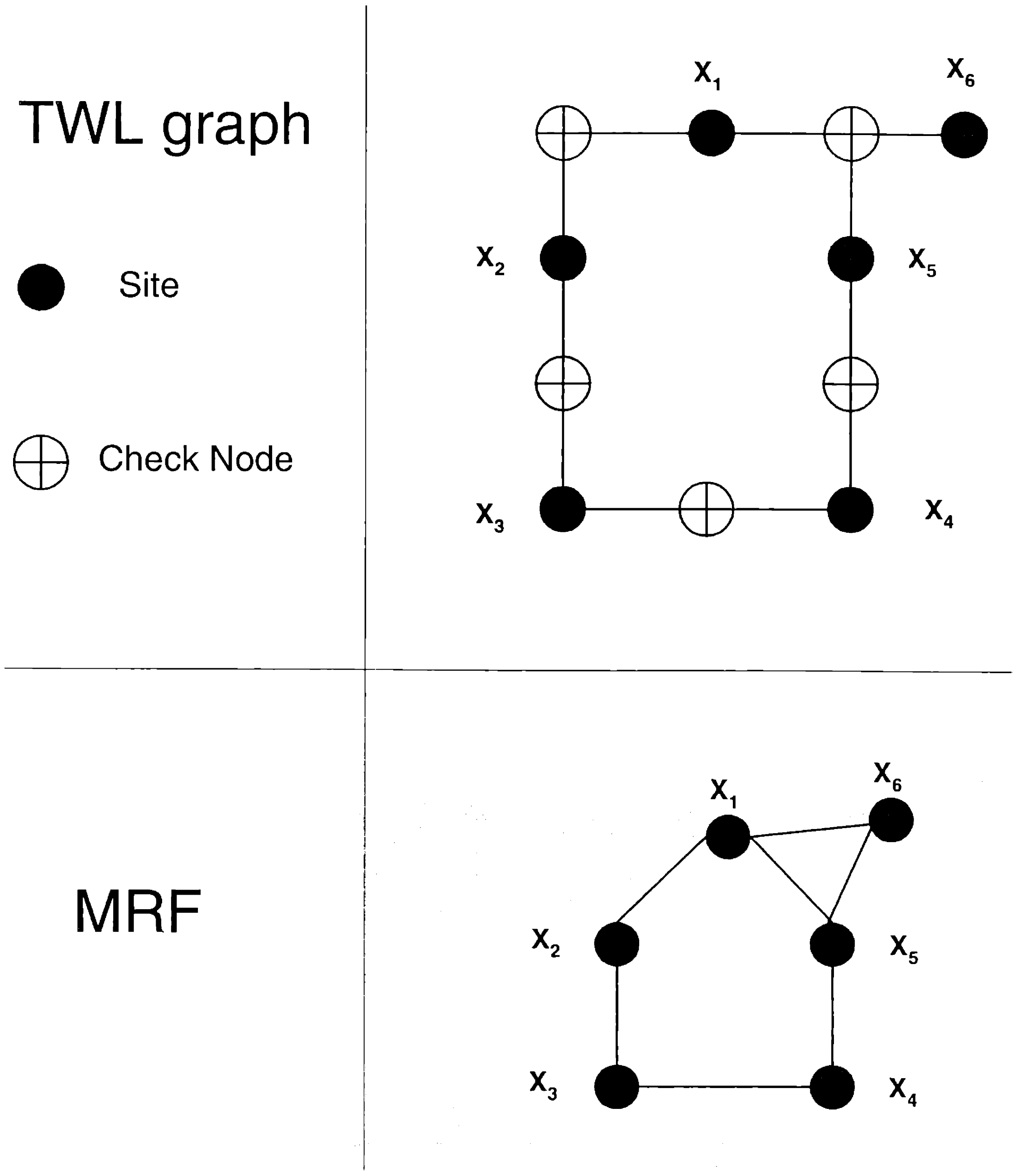

Figure 2.2 Example of a TWL graph and the corresponding MRF.

names.

Theorem 1 A MRF constructed from a TWL graph according to the procedure described above satisfies **(G)**.

Proof We prove this by a series of lemmas.

Lemma 1 Let A and B be two non-adjacent sets of sites on a TWL graph, and let S separate A and B, where S is disjoint from A and B. Then for any $\alpha \in A$ and any $\beta \in B$, it holds that $\alpha \perp \beta | S$.

Proof By definition, S must include at least one site in every path from every site in A to every site in B. Let us choose $S' \subset S$ as a *minimal separator set*; i.e., S' includes one and only one site from every path from every site in A to every site in B, and no other sites.

Let us extend A to A' in the following fashion:

- A' will contain A
- A' will contain any site not already in A' or S' that is adjacent to some site in A'.

We extend B to B' similarly.

We have now divided the TWL graph into four disjoint sets of nodes:

- Nodes in S'.
- Nodes in A'.
- Nodes in B'.
- Nodes that have no paths to either A or B. This set of nodes is irrelevant, as none of these nodes is connected to the section of TWL graph that we are considering. For completeness, these nodes can be associated with A'.

The TWL graph is thus separated into three disjoint sets of nodes: S', A', B'. Moreover A' and S' are adjacent, B' and S' are adjacent, and A' and B' are non-adjacent, with S' separating A' and B'.

Recall that any site on a TWL graph is completely determined by its adjacent sites; i.e., the value of any site is a deterministic function of the values of sites adjacent to it.

We now consider any site $\alpha'' \in A'$ that is adjacent to at least one site $\sigma \in S'$. We can then write $\alpha'' = f(A'', S'')$ where A'' is the set of all sites in A' adjacent to α'' and S'' is the set of all sites in S' adjacent to α''. Given all the nodes in S', all nodes in A' depend only on nodes in A' since A' is adjacent only to S'. Therefore, given S', A' is independent of B', and vice versa.

Since $A \subset A'$ and $B \subset B'$ and $S' \subset S$, it follows that $A \perp B | S$.

□

Note that the above statement is the equivalent of **(G)** for TWL graphs. Thus all deterministic TWL graphs have the Global Markov Property.

Lemma 2 A path from α to β exists on an MRF if and only if the same path existed on the original TWL graph.

Proof Every path on the TWL is preserved on the MRF since all adjacent sites on the TWL graph become adjacent vertices on the MRF by construction.

Suppose now that a new path was created by a transformation. This means that some vertices which were not adjacent on the TWL graph are adjacent on the MRF, which is impossible by construction.

□

Lemma 3 A set of vertices S on an MRF separates A and B if and only the corresponding set of sites separates A and B on the original TWL graph, and *vice versa.*

Proof If S separates A and B on the MRF, then it must include at least one node on every path from every node in A to every node in B. Since (by Lemma 3) there is one-to-one correspondence between paths on the MRF and paths on the TWL graph, the corresponding set of sites on the TWL graph must include at least one site on every path from every site in A to every site B. This set is by definition a separator.

The converse is proven analogously. □

Concluding the Proof of Theorem 1 By Lemma 3, if S separates A and B then $A \perp B | S$ on both the TWL graph and the MRF obtained from it. Therefore (G) holds on this MRF. This concludes the proof of Theorem 1.

□

2.4 Decomposable graphs

A subset C of nodes V of an MRF $\mathcal{G}$ is called a *complete subset* if all nodes of C are connected to each other. Complete subsets that are maximal are called *cliques.*

A triple (A, B, C) of disjoint subsets of V is said to decompose $\mathcal{G}$ if:

i $V = A \cup B \cup C$.

ii $A \perp B | C$.

iii C is a complete subset of V.

Since we are interested in graphs in which all the nodes are discrete (since all the alphabets are finite), the definition above matches the definition of strong decomposability in [Lau96].

A *triangulated graph* is a graph in which every cycle of four nodes or more contains a chord. A triangulated graph appears to be split into triangles, hence the name. *Triangulation* is the procedure of adding chords to a graph so that a triangulated graph results.

Note that addition of chords does not violate the underlying probability distribution which the graph represents. A chord between two nodes means that there is a direct dependence between these two variables. By direct dependence we mean that given all the other variables in the distribution, the two variables in question still exhibit a dependence on each other. The dependence that the additional chords model is "the two variables are independent given the rest of the graph." However, by adding the extra chords we lose some information about the special structure of our particular distribution.

Lauritzen [Lau96] proves that the following statements are equivalent:

i $\mathcal{G}$ is decomposable.

ii $\mathcal{G}$ is triangulated.

iii Every minimal separator set between any two nodes α and β is complete.

2.5 Factorizable graphs

A distribution P on $\mathcal{X}$ is said to *factorize* according to $\mathcal{G}$ if for all complete subsets $\Lambda \subset V$, there exist non-negative functions Ψ_Λ that depend only on the variables in Λ such that P can be written as:

$$P = \prod\nolimits_{all\ complete\ \Lambda} \Psi_\Lambda(x) \tag{2.1}$$

The functions Ψ_Λ are called *potential functions* or simply *potentials*. These are not uniquely determined. Since cliques are maximal complete subsets, we may assume them to be functions over the cliques of V. Equation (3.1) then becomes:

$$P = \prod\nolimits_{all\ cliques\ C} \Psi_C(x) \tag{2.2}$$

where Ψ_C are potentials over cliques.

If P factorizes, then P is said to have the property **(F)**. Lauritzen [Lau96] shows that **(F)** $\Rightarrow$ **(G)** $\Rightarrow$ **(L)** $\Rightarrow$ **(P)** for any MRF. The converse, however, is not always true. Lauritzen [Lau96] proves that **(F)** $\Leftrightarrow$ **(G)** $\Leftrightarrow$ **(L)** $\Leftrightarrow$ **(P)** if the distribution P is continuous and strictly positive, and comments that the continuity restriction can probably be relaxed, but that the positivity of the distribution is essential.

The distributions that arise in coding contexts are discrete and contain many zero values. Therefore, we need to show that these distributions are indeed factorizable. Theorem 1 in Chapter 2 already demonstrated that for all distributions that can be modeled by TWL graphs, and thus all distributions that arise in coding, it holds that

(G) $\Leftrightarrow$ **(L)** $\Leftrightarrow$ **(P)**. However, it is still not established whether for such a distribution **(F)** $\Leftrightarrow$ **(G)** will hold.

In order for **(F)** $\Leftrightarrow$ **(G)** to hold for any distribution, the graph must be decomposable. If our graph is decomposable, and if the distribution obeys **(G)** over the graph, then [Lau96] provides a way for us to factorize the distribution over this graph.

Theorem 2 Assume that (A, B, S) decompose $\mathcal{G}$. Then a probability distribution P factorizes with respect to $\mathcal{G}$ if and only if both its marginal distributions $P_{A \cup S}$ and $P_{B \cup S}$ factorize with respect to $\mathcal{G}_{A \cup S}$ and $\mathcal{G}_{B \cup S}$ respectively, and if the densities satisfy:

$$P(x)P_S(x_S) = P_{A \cup S}(x_{A \cup S})P_{B \cup S}(x_{B \cup S}) \tag{2.3}$$

Proof: Given in [Lau96].

We note that **(2.3)** is a re-statement of the Global Markov Property, since it can equivalently be written as:

$$P(x|x_S) = P_{A \cup S}(x_{A \cup S}|x_S)P_{B \cup S}(x_{B \cup S}|x_S) \tag{2.4}$$

where a multiplication of **(2.4)** by $(P_S(x_S))^2$ results in **(2.3)**.

Thus, we conclude that any graph obeying the Global Markov Property is either decomposable or can be made decomposable by triangulation.

Next we note that **(2.3)** implies a recursive procedure of factorization down to cliques, which are decomposable trivially into themselves. Then **(2.3)** can be re-written as

$$P(x) \prod\nolimits_{all\ minimal\ clique\ separator\ sets\ S} P_S(x_S) = \prod\nolimits_{all\ cliques\ C} P_C(x_C) \tag{2.5}$$

or

$$P(x) = \frac{\prod_C P_C(x_C)}{\prod_S P_S(x_S)} \tag{2.6}$$

Note that **(2.6)** has the form of **(2.2)**.

Consider now a graph that decomposes into two cliques A and B with a minimal separator set S. We note that this implies that $S \subset A$ and $S \subset B$. If the joint distribution over the whole graph is to be valid, it is only necessary that the marginal distribution of S found by marginalizing P_A be the same as the marginal distribution of S found by marginalizing P_B, i.e. we need

$$P_S = \sum_{A\setminus S} P_A = \sum_{B\setminus S} P_B \tag{2.7}$$

Given two distributions that do not satisfy this constraint, it is possible to find a joint distribution that does simply by performing the following sequence of operations:

$$\phi_S^* = \sum_{A\setminus S} P_A \tag{2.8a}$$

$$\Psi_B^* = P_B \cdot \frac{\phi_S^*}{P_S} \tag{2.8b}$$

$$\phi_S^{**} = \sum_{B\setminus S} \Psi_A \tag{2.8c}$$

$$\Psi_A^{**} = P_A \cdot \frac{\phi_S^{**}}{\phi_S^*} \tag{2.8d}$$

where multiplication and division are performed "variable by variable" and $\frac{0}{0}$ is defined as 0.

At this point the functions Ψ_A^{**}, Ψ_B^* and ϕ_S^{**} satisfy **(2.3)** and **(2.6)**. If we normalize them to become valid probability distributions, then we get a valid probability distribution over the whole graph:

$$P = \frac{P_A^{**} P_B^*}{P_S^{**}} \tag{2.9}$$

where P_A^{**}, P_B^* and P_S^{**} are normalized potential functions.

The operations described in **(2.8a-d)** preserve all relations between variables in A that are not in S and variables in B that are not in S (up to a possible scale factor). Thus, we have found a procedure that, at least for a pair of cliques, finds the correct distribution over the separator set without altering the relationships within sets of variables that are independent of each other, given the separator set.

This leads us to conclude that if the cliques of a decomposable graph can be arranged into a tree with some additional structure, then for a distribution obeying the Global Markov Property over this graph this procedure can be used to find the correct distribution over the graph.

In the next two sections we construct such a tree and describe a scheme which uses operations in **(2.8a-d)** to find the correct joint probability distribution on the whole graph, given some initial distributions on the cliques which may not be consistent. The existence of such a tree is implied by the fact that the distribution is factorizable, as this tree will be simply a graphical representation of **(2.6)**.

2.6 The junction tree

A *junction tree* for a graph $\mathcal{G}$ is formed by taking the cliques to be the vertices and the minimal separator sets to be the edges. It is clear that if all the possible edges are used, the resulting "graph" (formally known as a *hypergraph*) is likely to contain cycles. Thus, one chooses only enough edges to build a tree.

Thus, for a given graph, a number of different junction trees can be constructed. A specific junction tree is said to have the *Running Intersection Property* **(R)** if for every pair of cliques C_1 and C_2 containing a common node c, every clique in the path $\{C_1, C_2\}$ in the junction tree contains c as well.

Figures 2.3 and 2.4 show examples of MRF's. The MRF in Figure 2.3 is already triangulated. The MRF in Figure 2.4 is triangulated by addition of edges indicated using dashed lines. Below them, junction trees corresponding to the MRF's are shown. For the MRF in Figure 2.3 we show a junction tree with **(R)**. A junction tree without **(R)** cannot be built for this MRF. For the MRF in Figure 2.4 we show two junction trees, one with **(R)** and one without **(R)**.

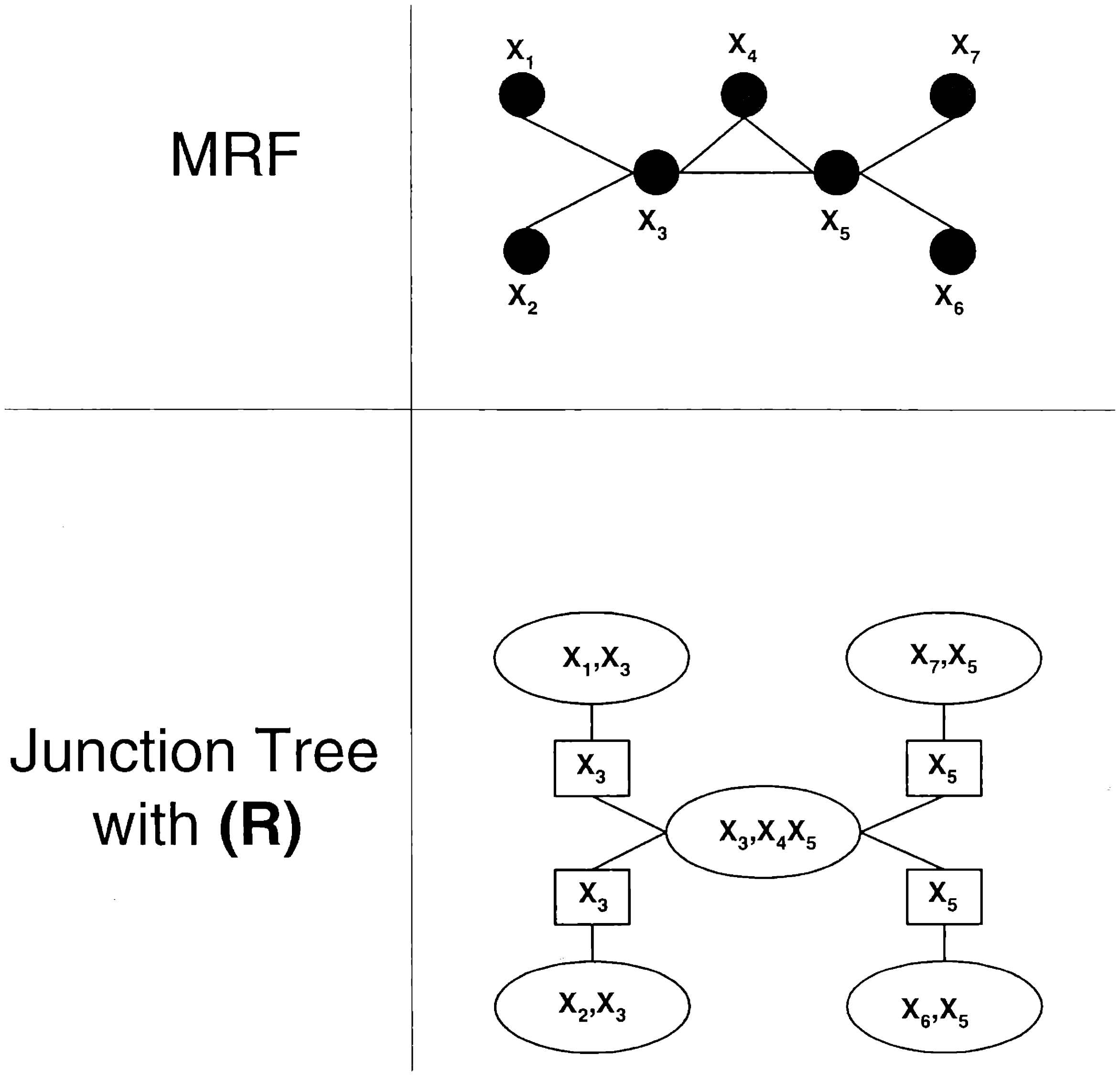

Figure 2.3 Example of an MRF and corresponding junction tree.

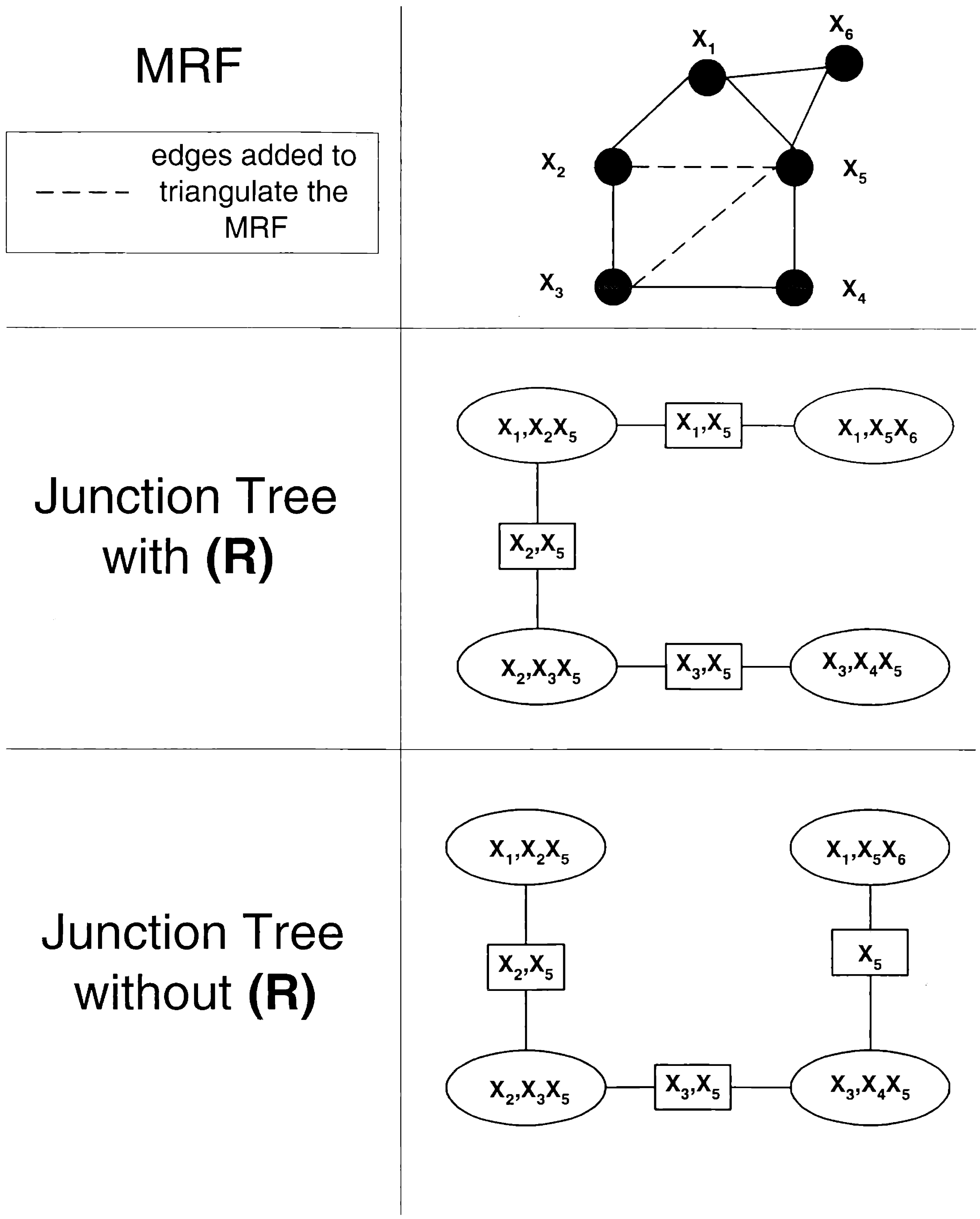

Figure 2.4 Example of an MRF and corresponding junction trees.

2.7 The LS procedure for exact probability propagation

We now present the procedure for exact probability propagation that is developed in [LS88].

Call the operation described by **(2.8a)** and **(2.8b)** *passing a message from A to B*, where S is the content of the message, or simply *the message*. Similarly, **(2.8c)** and **(2.8d)** represent passing a message from B to A, where S is again the message.

The following scheme is based on [LS88] and can also be found in [Jen96]. It ensures that the whole junction tree is updated in time at most $2DT$, where D is the diameter of the junction tree, and T is the time it takes to pass one message.

1. Start at the leaf nodes and let each one pass a message.

2. A clique A can send a message to its neighbor B only if all other neighbors of A have already sent A a message, and A has not yet sent a message to B. It does not matter whether B has sent A a message or not.

3. The procedure terminates when every pair of cliques has exchanged messages once in each direction (i.e. once from A to B and once from B to A.) At this point the whole graph is updated.

In the coding context we can make the following notes about this procedure:

1. We usually do not care about normalizing the final result to a proper probability distribution.

2. A channel observation can be incorporated as a message coming from the outside into one of the cliques on the junction tree that contains that particular node. It is handled as any other message would be.

3. Since the complexity of the algorithm depends on the time it takes to pass a message, which depends on its size, we would like to keep the messages as small

as possible, preferably representing a single variable. We now show under what conditions this will occur.

Theorem 3. The following are true if and only if the original TWL graph is a tree.

1. All cliques in the resulting MRF correspond to check nodes in the TWL graph and vice versa.
2. The resulting MRF is already triangulated.
3. Each edge of the junction tree of the resulting MRF corresponds to exactly one node.
4. There is only one possible junction tree that can be constructed. It has the Running Intersection Property.

Proof

We first prove the direct statement. If the original TWL graph is a tree, then:

1. By construction (see Section 2.3) every check node results in a clique, since all sites connected to the same check node are connected to each other.

   Suppose an additional clique appeared. This implies that there is at least one connection between sites that are not connected to the same check node. This is impossible since, by construction, sites can be connected only to sites at the same check node.

2. Other than the cliques, the MRF has no cycles, since the original TWL graph had no cycles. Cliques by definition cannot contain cycles of four or more nodes with no chords. Thus the graph is triangulated.

3. Since no additional edges were added to the graph in order to triangulate it, the only nodes that the cliques share correspond to sites that were shared by check nodes in the original TWL graph. Since the original TWL graph was cycle-free,

only single sites were shared between check nodes. Thus, at most one node is shared by cliques and each edge on the junction tree consists of precisely one variable.

4. Since the original TWL graph was a tree, we need to use all possible connections to make the junction tree. This implies uniqueness of this junction tree. Suppose this tree does not have **(R)**. This means that there are two cliques that share some node, say a, that are not directly connected. It follows that the check nodes on the TWL graph that correspond to these cliques were not connected, which implies two sites on the TWL graph corresponding to a, which is impossible.

To prove the converse we consider three different cases:

I. Suppose the original TWL graph contains a cycle of length two. Then there is a pair of check nodes that are connected to two sites, which means that there is a pair of cliques on the resulting MRF which share two nodes, which means the edge on the junction tree connecting these two cliques will consist of two nodes and Statement 3 does not hold.

II. Suppose the original TWL graph contains a cycle of length three. Then, by construction, the nodes on the MRF corresponding to the sites in this cycle must be part of a clique (they are all connected to each other). However, this clique obviously does not correspond to any check nodes on the original TWL graph and Statement 1 does not hold.

III. Suppose the original TWL graph contains a cycle of length more than three without a chord. Then, by construction, a cycle of length more than three must appear in the MRF as well. Then the MRF is not triangulated and Statement 2 does not hold. If the cycle has a chord, consider the smaller cycles induced by the chord. Either this proof or the proof as in **II** above must apply.

□

Theorem 3 implies that if we start with a cycle-free TWL graph, then the LS procedure will be identical to the sum-product algorithm as described in [Wib96], [WLK95].

If the original TWL graph contains cycles, then [Wib96], [WLK95] show that the sum-product algorithm may not be optimal; in general, it is not even guaranteed to converge. The LS procedure will still be optimal. However, its complexity will increase, since the size of the message depends on the number of variables that an edge of the junction tree consists of. The increase is linear in "time to process a message." Unfortunately the "time to process a message" is exponential in the number of variables that the message consists of; thus the overall increase will be exponential.

In fact, we now show that the LS procedure is equivalent to using the sum-product algorithm on a cycle-free TWL graph representation of the given code. Thus its complexity is lowerbounded by complexity of the sum-product algorithm on the minimal cycle-free TWL graph of a particular code. We first illustrate this using a simple example of a graph with a single cycle: namely a tail-biting trellis. Then we show this in the general case.

2.8 The LS procedure on tail-biting trellises

A tail-biting trellis (TBT) is defined similarly to a conventional trellis. The difference is that, while in a conventional trellis the end nodes correspond to trivial one-valued state spaces, this restriction is relaxed for a TBT. For a TBT the end state spaces can be multi-valued, and it is required only that the end state be the same as the starting state. Graphically, this can be represented by making the starting and ending states correspond to the same node. This turns the time axis of the conventional trellis into a cycle (the trellis "bites its tail"). An example of a conventional trellis and a TBT of the same length is shown in Figure 2.5.

The main advantage of using a TBT representation of a block code, as opposed to a conventional trellis, is the reduction of the size in the state space, as pointed out in [Wib96] and [CFV98]. Suppose a minimal conventional trellis for some block code

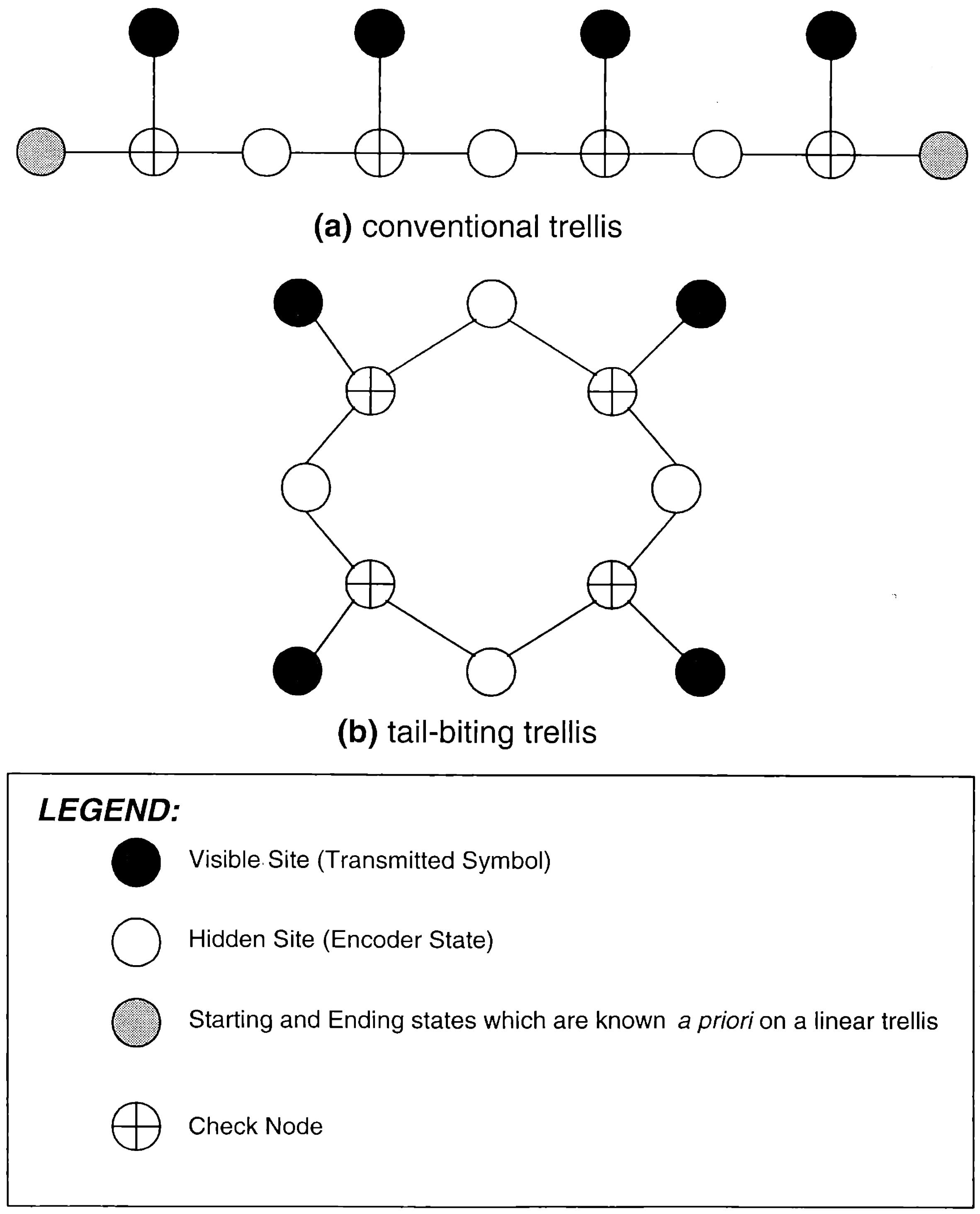
(a) conventional trellis
(b) tail-biting trellis
LEGEND:
Visible Site (Transmitted Symbol)
Hidden Site (Encoder State)
Starting and Ending states which are known *a priori* on a linear trellis
Check Node

Figure 2.5 Example of a conventional trellis and a TBT of length 4.

has a state space size S (i.e. in at least one place on the trellis we have S possible states). Then a TBT for the same code has to have state space size at least $\sqrt{S}$. In practice, the lower bound of $\sqrt{S}$ is often met.

A good discussion of TBTs is given in [CFV98]. Further details of this subject are not important for the discussion in this chapter. We only note that for any linear code a TBT can be constructed.

Consider the complexity of the min-sum algorithm on the conventional trellis of some block code. Say this trellis has state space size S. The min-sum algorithm will be optimal here (in fact, it will be the bidirectional Viterbi algorithm). Its complexity is roughly proportional to S.

Now consider a TBT for this code with state space size S'. It is easy to show that any junction tree with **(R)** of the loop will involve edges which consist of two nodes. (It is, of course, possible to make junction trees with even "thicker" edges, but we wish to find junction trees with minimum edge thickness.) Thus, the complexity of the LS procedure will be exponential in $(S')^2$. Since $S' \geq \sqrt{S}$ we see that the LS procedure is at least as complex as using a conventional trellis for the same code. Since in practice S' is often equal to $\sqrt{S}$, the two exact procedures often have approximately the same complexity.

2.9 Junction trees and the cut-set bound

We now consider general graphs and show that constructing a junction tree with **(R)** is essentially equivalent to constructing a cycle-free TWL graph for a given code using the "cut" method introduced in [Wib96]. This will imply that the complexity of the LS procedure is lowerbounded by the "cut-set bound" of [Wib96].

We note that a *cut* on a TWL graph is defined in Chapter 7 of [Wib96] as:

> a partition (I, K, J) of the sites such that no check set contains sites from both I and J.

This definition implies that in the MRF of the code, K becomes a separator set for I and J. All that is needed for the MRF to be decomposable is that the separator set be complete (i.e. all nodes of the separator set be connected to each other). Triangulation of the MRF assures us of this property.

We next note that [Wib96] uses cut sets to construct cycle-free graph representations of codes. This construction is equivalent to finding a junction tree with **(R)** for a given MRF. Therefore we can draw the following conclusions:

- Finding a junction tree with **(R)** for an MRF of a code is equivalent to finding a cycle-free TWL graph for the same code.
- The complexity of the LS procedure is lowerbounded by the complexity of the sum-product algorithm on a minimal cycle-free TWL graph of the given code. This is the "cut-set bound" of [Wib96]. The complexity of the LS procedure is thus lowerbounded by the cut-set bound.

Since minimal cycle-free TWL graphs of powerful codes are known to be very complex, we are motivated to study iterative algorithms (min-sum as well as sum-product) on TWL graphs with cycles. Although sub-optimal, these have been experimentally shown to do very well. For example, with turbo codes, the sum-product algorithm achieves performance within 0.5 dB of the Shannon limit. These algorithms also have relatively low complexity. The next two chapters present some results on the performance of the min-sum algorithm on TWL graphs with cycles.

Chapter 3

Performance of the min-sum algorithm

In the last chapter we found that the exact approach to decoding on graphs results in a very complex procedure for most practical codes. An alternative that suggests itself is to examine the performance of min-sum and sum-product algorithms on the TWL graphs with cycles "as is." We know from [Wib96] and [WLK95] that these algorithms will not in general be optimal. Moreover, these references say nothing about convergence. However, it is possible to say more about the behavior of these algorithms on graphs with special structure.

The approach to this problem taken by several researchers in the field has been to start by examining the simplest graph that contains a cycle, namely a graph with a single loop. Their motivation was that analysis of this simplest case could possibly lead to generalizations which would be helpful in understanding more complex codes.

A tail-biting trellis (TBT), which is a way of representing block codes, was introduced in Section 2.8 as a TWL graph which has a single cycle. A significant amount of recent research has been devoted to this special case.

Anderson and Hladik [AH98] and others have proved convergence of the sum-product algorithm on a TBT. Experimentally, [AH98] found that convergence occurs very fast. We have obtained similar results in simulating the sum-product algorithm on the TBT of the (24, 12, 8) Golay code, to be presented in Chapter 4. However, while [AH98] predicts what the algorithm will converge to, it does not establish the relationship between the output of the algorithm and the correct *a posteriori* probabilities. Weiss [Wei97] shows that for binary codes the algorithm will make optimal decisions.

The workings of the min-sum algorithm on a TBT have been analyzed in [FKV97], [Wib96] and [FKM98], where it is shown that the algorithm works over the "unwrapped" TBT which is a bi-infinite conventional trellis. The min-sum algorithm is optimal on the conventional trellis; however, it can converge to a codeword of the trellis that is not a valid codeword of the original code. These invalid codewords have been called "pseudocodeword" in [FKV97], and were first observed by Agrawal [Agr97].

Recently, Frey, Kötter and Vardy [FKV97] attempted to explain the effect pseudocodewords have on the performance of the min-sum algorithm. In this chapter we build on their work and define a measure of probability of error to a pseudocodeword when AWGN channel is used for transmission. We first derive this result for a tail-biting trellis and demostrate some useful lower and upper bounds. We then generalize this result and show that it is equivalent to a result obtained previously by Wiberg in Chapter 6 of [Wib96].

3.1 Pseudocodewords

Pseudocodewords appear when the min-sum algorithm is used on a TBT because the algorithm, in looping around multiple cycles, can find a path that is periodic with a period of more then one cycle. For a precise definition, we first have to introduce the concept of a computation tree for decoding of a code defined on a graph.

A *computation tree* is a tree on which the algorithm used (min-sum in our case) performs exactly the same computations as it does on the original graph. We note the following things about a computation tree:

- A computation tree is dependent on a particular graphical representation of a code and on the computation schedule on that graph. A different graphical representation or schedule will yield a different computation tree. For example, the computation tree for a conventional trellis, which is already a tree, is the trellis itself. The computation tree for a TBT is an "unwrapped" bi-infinite TB trellis. These two trees are in general different, even when the conventional trellis and the TBT represent the same code.

- A computation tree is infinite if the algorithm is not guaranteed to converge in finite number of iterations. The only known graphs that have finite convergence are finite trees.

- Whereas in the original graph there is only one site for each variable, in a computation tree each variable may have many sites corresponding to it.

Sometimes it is also helpful to talk of finite computation trees that result when the algorithm is stopped at a particular point. We will use the term *computation tree* for this situation as well.

The root of a computation tree is the site at which the algorithm is started. If we consider an infinite computation tree, then the root is usually not important. If we consider a finite computation tree for the case when the algorithm is stopped before it has converged, then both the location of the root and the stopping time (i.e., the depth of the computation tree) may be important.

The computation tree of a TB trellis for any reasonable decoding schedule is an "unwrapped" bi-infinite trellis, as illustrated in Figure 3.2. Figure 3.3 provides another illustration of a TWL graph and its computation tree, taken from [FKV97].

A computation tree is said to be *balanced* [Wei97] if each variable has the same fraction of nodes on the tree corresponding to it. The computation tree in Figure 3.2 is balanced, because the sequence of sites on the trellis is periodic, and each site occurs precisely once per period. However, the computation tree in Figure 3.3 is not balanced. This is shown in [FKV97], where the multiplicities of the three sites are computed using a recursion formula; asymptotically x_2 and x_3 appear 0.9156 times as often as x_1.

Let $\mathcal{C}$ be some code, and let $\mathcal{C}^{\mathcal{T}}$ be the code described by the computation tree of $\mathcal{C}$.

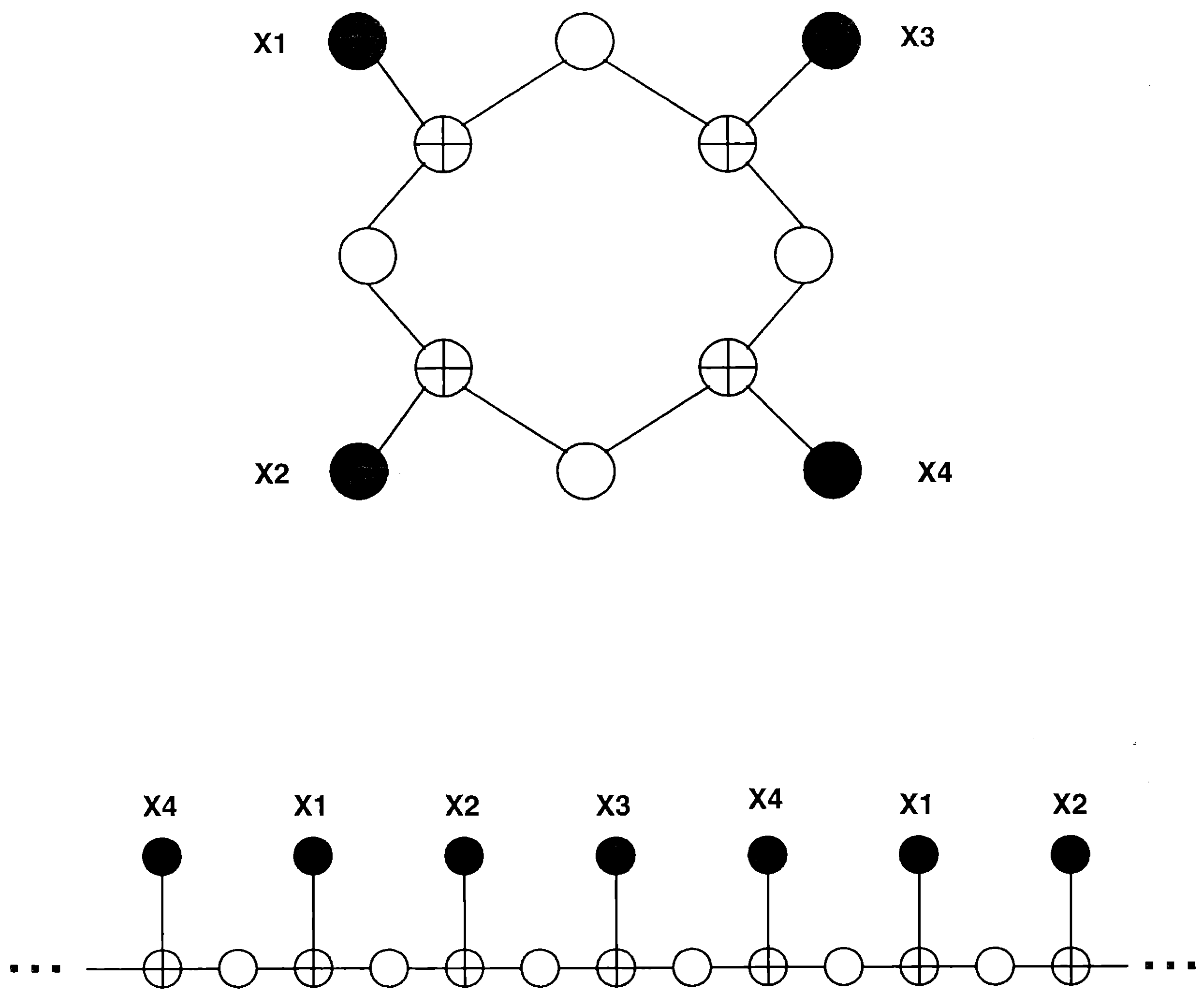

Figure 3.1 A tail-biting trellis with its bi-infinite computation tree.

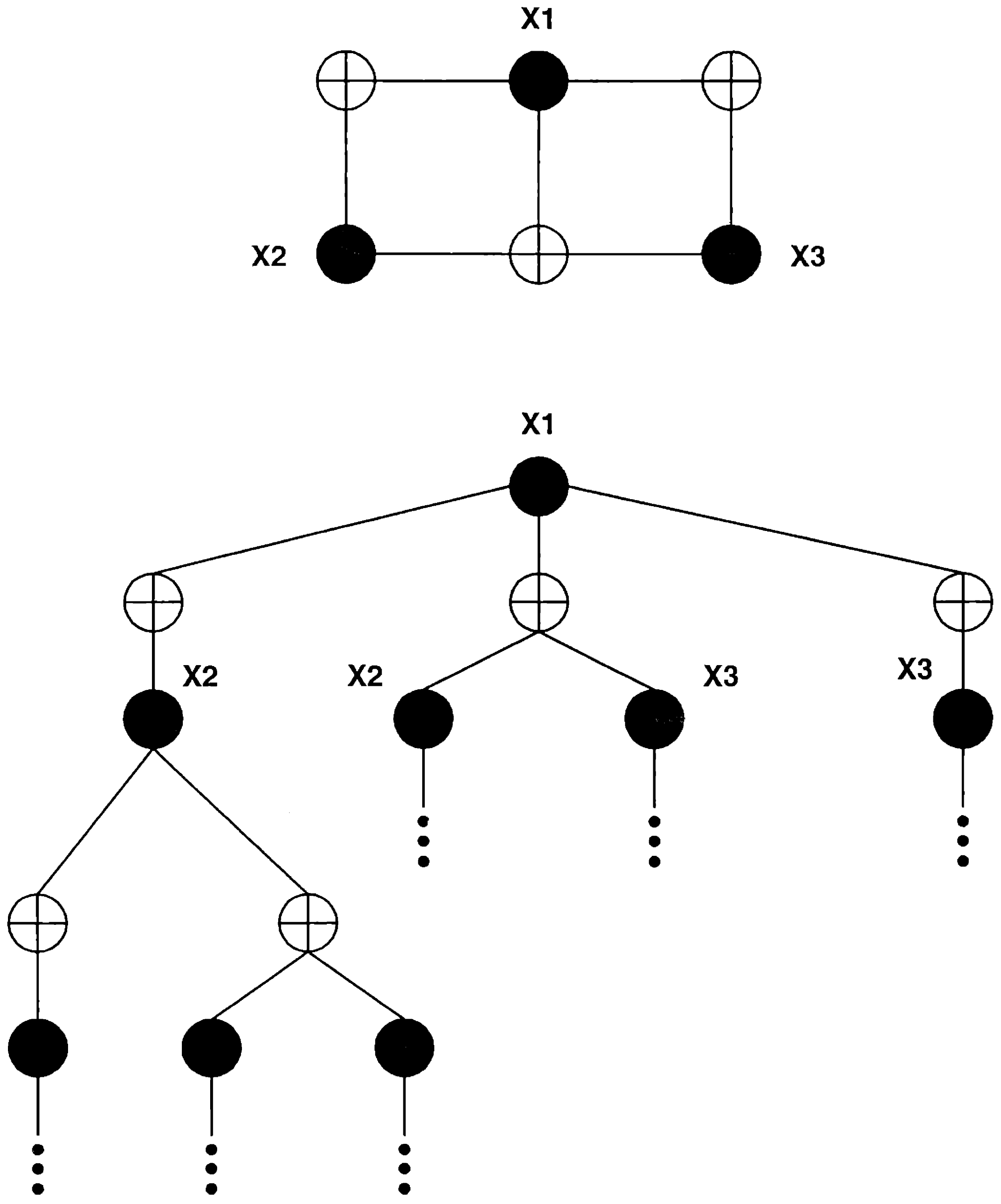

Figure 3.2 A TWL graph and its computation tree with the algorithm started at X_1.

A codeword of $\mathcal{C}$, repeated an appropriate number of times, may also be a codeword of $\mathcal{C}^{\mathcal{T}}$; however, there may be some codewords of $\mathcal{C}^{\mathcal{T}}$ which are not codewords of $\mathcal{C}$. Collectively the codewords of $\mathcal{C}^{\mathcal{T}}$ are referred to as *pseudocodewords* [FKV97].

We note that the definitions above apply to TWL graphs in general, not just to TBTs. Thus we expect pseudocodewords that are not valid codewords to appear for all graphs that have computation trees different from the graph itself.

3.2 The effective weight of a pseudocodeword for an AWGN channel

We consider binary codes, i.e., codes where each symbol takes values from the binary alphabet $\{0, 1\}$. We assume that binary antipodal signaling is used.

The probability of error to a codeword when using an ML decoder for a linear code on an AWGN channel depends on the Hamming distance d of this codeword from the transmitted codeword. If we restrict ourselves to linear block codes, then it is sufficient to consider the case where the all-zero codeword is transmitted. The performance of an ML decoder then depends on the Hamming weights of the codewords, and in the high-SNR limit it is determined by the minimum-weight codewords. The pairwise probability of error to a codeword when the all-zero codeword is transmitted is given by:

$$Pr(error) = Q\left(\sqrt{d \cdot Rate \cdot \frac{2E_b}{N_0}}\right) \tag{3.1}$$

Let $\mathbf{x^P}$ be a codeword of $\mathcal{C}^{\mathcal{T}}$ and $\mathbf{x}$ be a codeword of $\mathcal{C}$. Frey, Kötter and Vardy [FKV97] use the fact that the min-sum algorithm is optimal for $\mathcal{C}^{\mathcal{T}}$ to show that it will decode to:

$$\hat{\mathbf{x}}^{\mathbf{P}} = \arg \max_{\mathbf{x^P} \in \mathcal{C}^{\mathcal{T}}} < \mathbf{y}, \mathbf{l}(\mathbf{x^P}) > \tag{3.2}$$

where $\mathbf{y}$ is a vector of channel observations (as log-costs), $l_i(\mathbf{x^P})$ is the number of times that the bits of $\mathbf{x^P}$ corresponding to the i^{th} bit of $\mathbf{x}$ are equal to 1, and $< \mathbf{y}, \mathbf{l}(\mathbf{x^P}) >$

is the inner product $\sum_i y_i l_i(\mathbf{x^P})$.

The definition of $l_i(\mathbf{x^P})$ makes sense in the context of [FKV97] under the restriction to a finite computation tree, which corresponds to stopping the min-sum algorithm at some point. Alternately, the definition makes sense under the assumption that the min-sum algorithm converges on a given graph and we can restrict our attention to pseudocodewords that the min-sum algorithm has converged to.

If we are dealing with a balanced computation tree, then we can replace $l_i(\mathbf{x^P})$ by the normalized fraction $f_i(\mathbf{x^P}) = l_i(\mathbf{x^P})/m_i$, where m_i is the total number of sites in $\mathcal{C}^\mathcal{T}$ that correspond to the i^{th} bit of $\mathcal{C}$. Equation (**3.2**) then becomes:

$$\hat{\mathbf{x}}^\mathbf{P} = \arg \max_{\mathbf{x^P} \in \mathcal{C}^\mathcal{T}} < \mathbf{y}, \mathbf{f}(\mathbf{x^P}) > \tag{3.3}$$

We know that the min-sum algorithm will converge on a TBT [Wei97], [FKM98]. This is clear from the fact that the min-sum algorithm is nothing but the bidirectional Viterbi algorithm running on the unwrapped trellis. We conclude that since the computation tree for a TBT is balanced, the min-sum algorithm for a TBT will converge to $\arg\max_{\mathbf{x^P} \in \mathcal{C}^\mathcal{T}} < \mathbf{y}, \mathbf{f}(\mathbf{x^P}) >$.

We now proceed to compute the probabiity of error from the all-zero codeword to a pseudocodeword. In the discussion that follows we assume that binary antipodal signaling is used on an AWGN channel and that the all-zero codeword is sent, corresponding to channel input of $-\sqrt{S}$, where $S = Rate \cdot \frac{2E_b}{N_0}$. Therefore, the y_i's are distributed according to $\mathcal{N}(-\sqrt{S}, 1)$ and are independent, given our knowledge that the all-zero codeword was sent. The correlation $< \mathbf{y}, \mathbf{f}(\mathbf{x^P}) >$ is equal to 0 for the all-zero codeword, because for an all-zero codeword $f_i(\mathbf{x^P}) = 0$ for all i. Therefore, an error occurs when another $\mathbf{x^P}$ is found such that $< \mathbf{y}, \mathbf{f}(\mathbf{x^P}) >$ is greater than 0.

Before deriving the general formula, we investigate the two simplest special cases.

Consider the probability of error to a valid codeword. In such a case $f_i(\mathbf{x^P})$ is constrained to take values in $\{0, 1\}$. Then we have:

$$< \mathbf{y}, \mathbf{f}(\mathbf{x^P}) > = \sum_{i:x_i=1} y_i \tag{3.4}$$

Let E_1 be the number of bits of $\mathbf{x}$ that are equal to 1. Then the random variable $z = <\mathbf{y}, \mathbf{f}(\mathbf{x}^{\mathbf{P}})>$ has the distribution $\mathcal{N}(-E_1\sqrt{S}, E_1)$. The probability of error from the all-zero codeword to this codeword is $Q(\sqrt{E_1 S})$, which is the standard result for a valid codeword.

Consider now a pseudocodeword with a period equal to 2 cycles of the tail-biting trellis. The bits of $\mathbf{x}^{\mathbf{P}}$ corresponding to the i^{th} bit of $\mathbf{x}$ can take on three possible combinations: either all are 0, or all are 1, or exactly half are 0 and half are 1. Thus $f_i(\mathbf{x}^{\mathbf{P}})$ takes values in $\{0, \frac{1}{2}, 1\}$. We can then write:

$$<\mathbf{y}, \mathbf{f}(\mathbf{x}^{\mathbf{P}})> = \left(\sum_{i:x_i=1} y_i\right) + \frac{1}{2}\left(\sum_{i:x_i=\frac{1}{2}} y_i\right) = z_2 + z_1 = z \tag{3.5}$$

Let $E_2 = \left|\{i : f_i(\mathbf{x}^{\mathbf{P}}) = 1\}\right|$ and $E_1 = \left|\{i : f_i(\mathbf{x}^{\mathbf{P}}) = \frac{1}{2}\}\right|$. Then the random variable z_2 is distributed according to $\mathcal{N}\Big(-E_2\sqrt{S}, E_2\Big)$, and the random variable z_1 is distributed according to $\mathcal{N}\Big(-\frac{E_1}{2}\sqrt{S}, \frac{E_1}{4}\Big)$. The random variable z is distributed according to $\mathcal{N}\Big(-(E_2 + \frac{E_1}{2})\sqrt{S}, E_2 + \frac{E_1}{4}\Big)$. The probability that z is greater then 0 is given by:

$$Pr\{z > 0\} = Q\left(\sqrt{\frac{(E_2 + \frac{E_1}{2})^2}{E_2 + \frac{E_1}{4}} S}\right) \tag{3.6}$$

We note in **(3.6)** that the expression $\frac{(E_2+\frac{E_1}{2})^2}{E_2+\frac{E_1}{4}}$ takes the place of the weight of a codeword, and we define it to be the *effective weight* of a 2-cycle pseudocodeword. If $E_1 = 0$, the effective weight is equal to the average Hamming weight.

Let us now consider the general case of an M-cycle pseudocodeword. $f_i(\mathbf{x}^{\mathbf{P}})$ can take values in $\{0, \frac{1}{M}, \frac{2}{M}, \ldots \frac{M-1}{M}, \frac{M}{M}\}$. Define $E_j = \left|\{i : f_i(\mathbf{x}^{\mathbf{P}}) = \frac{j}{M}\}\right|$. In this case, we have:

$$<\mathbf{y}, \mathbf{f}(\mathbf{x}^{\mathbf{P}})> = \sum_{j=1}^{M}\left(\frac{j}{M}\sum_{i:x_i=\frac{j}{M}} y_i\right) = \sum_{j=1}^{M} z_j = z \tag{3.7}$$

where $z_j = \frac{j}{M}\sum_{i:x_i=\frac{j}{M}} y_i$.

Each z_j is distributed according to $\mathcal{N}\Big(-\frac{j}{M}E_j\sqrt{S}, \left(\frac{j}{M}\right)^2 E_j\Big)$. Therefore z will be distributed according to $\mathcal{N}\Big(-\sum_{j=1}^{M}\frac{j}{M}E_j\sqrt{S}, \sum_{j=1}^{M}\left(\frac{j}{M}\right)^2 E_j\Big)$. Then we have:

$$Pr\{z > 0\} = Q\left(\sqrt{\frac{\left(\sum_{j=1}^{M}\frac{j}{M}E_j\right)^2}{\sum_{j=1}^{M}\left(\frac{j}{M}\right)^2 E_j}S}\right) \tag{3.8}$$

We therefore define the effective weight of a pseudocodeword as:

$$W_{eff} = \frac{\left(\sum_{j=1}^{M}\frac{j}{M}E_j\right)^2}{\sum_{j=1}^{M}\left(\frac{j}{M}\right)^2 E_j} = \frac{\left(\sum_{j=1}^{M} jE_j\right)^2}{\sum_{j=1}^{M} j^2 E_j} \tag{3.9}$$

In fact, we can include in this formula the E_0 x_i's for which decode $f_i(\mathbf{x}^{\mathbf{P}}) = 0$ and obtain:

$$W_{eff} = \frac{\left(\sum_{j=0}^{M} jE_j\right)^2}{\sum_{j=0}^{M} j^2 E_j} \tag{3.10}$$

3.3 Bounds on effective weight

Recall that E_j is the number of bits of $\mathbf{x}$ which decode to $\frac{j}{M}$. Then the total number of ones in the pseudocodeword is equal to $\sum_{j=0}^{M} jE_j$. The average Hamming weight per cycle is $\frac{1}{M}\sum_{j=0}^{M} jE_j$.

If we have a valid codeword, then $E_j = 0$ for all $j \neq M$ and $j \neq 0$. The average Hamming weight per cycle then reduces to E_M, and $W_{eff} = \frac{(ME_M)^2}{M^2E_M} = E_M$ so the two are equal.

However, suppose that there exist some j's such that $0 < j < M$ and $E_j \neq 0$. Then we have:

$$\frac{1}{M}\sum_{j=0}^{M} jE_j = \frac{1}{M}\frac{\left(\sum_{j=0}^{M} jE_j\right)^2}{\sum_{j=0}^{M} jE_j} = \frac{\left(\sum_{j=0}^{M} jE_j\right)^2}{M^2E_M + \sum_{j=0}^{M-1} MjE_j} <$$
$$\frac{\left(\sum_{j=0}^{M} jE_j\right)^2}{M^2E_M + \sum_{j=0}^{M-1} j^2E_j} = \frac{\left(\sum_{j=0}^{M} jE_j\right)^2}{\sum_{j=0}^{M} j^2E_j} = W_{eff}$$

Thus, we see that W_{eff} is lowerbounded by the average Hamming weight per cycle of the pseudocodeword, and the lower bound is met with equality if and only if the pseudocodeword is a valid codeword of the original code $\mathcal{C}$.

We can also upperbound W_{eff}. Consider a pseudocodeword with no overlaps, so that $E_j = 0$ for all $j > 1$. Then $W_{eff} = \frac{E_1^2}{E_1} = E_1$; i.e., W_{eff} is equal to the total Hamming weight of the pseudocodeword.

We now show that in all other cases, W_{eff} is less then the total Hamming weight of the pseudocodeword. Assume that there is a $j > 1$ such that $E_j \neq 0$. Noting that the total Hamming weight is $\sum_{j=1}^{M} jE_j$, we have:

$$\sum_{j=1}^{M} jE_j = \frac{\left(\sum_{j=0}^{M} jE_j\right)^2}{\sum_{j=0}^{M} jE_j} > \frac{\left(\sum_{j=0}^{M} jE_j\right)^2}{\sum_{j=0}^{M} j^2E_j} = W_{eff}$$

Thus the total Hamming weight of the pseudocodeword is an upper bound and is achievable only for a pseudocodeword with no overlaps.

In general, for a set of pseudocodeword weights, the fewer overlaps there are, the higher will W_{eff} be.

3.4 Effective weight for general graphs

We now extend the concept of effective weight, W_{eff}, as defined above to general graphs. We assume that the graph of a code has a finite number of sites. Let N denote the length of a codeword in the original code $\mathcal{C}$. N is then also the number of visible sites in the original graph. We assume also that the pseudocodewords have a finite length, i.e., that the pseudocodewords are periodic with a finite period.

Let $l_i(\mathbf{x^P})$ be the number of times nodes in $\mathbf{x^P}$ corresponding to the i^{th} bit of the original codeword are equal to 1 over *one period* of the pseudocodeword. Since the computation tree may not be balanced, the set which $l_i(\mathbf{x^P})$ takes values in may be different for each i. We note that $\mathbf{l}(\mathbf{x^P})$ as used here is simply a scaled version of the $\mathbf{l}(\mathbf{x^P})$ used in [FKV97].

As no real simplification is possible, we consider $z = <\mathbf{y}, \mathbf{l}(\mathbf{x^P})> = \sum_{i=1}^{N} y_i l_i(\mathbf{x^P})$. z is then distributed according to $\mathcal{N}\left(-\left(\sum_{i=1}^{N} l_i(\mathbf{x^P})\right)\sqrt{S}, \sum_{i=1}^{N} l_i^2(\mathbf{x^P})\right)$

The probability of error from the all-zero codeword is then given by:

$$Pr\{z > 0\} = Q\left(\sqrt{\frac{\left(\sum_{i=1}^{N} l_i(x^T)\right)^2}{\sum_{i=1}^{N} l_i^2(x^T)}S}\right) \tag{3.11}$$

Therefore the effective weight is:

$$W_{eff} = \frac{\left(\sum_{i=1}^{N} l_i(x^T)\right)^2}{\sum_{i=1}^{N} l_i^2(x^T)} \tag{3.12}$$

This is precisely the "generalized weight" defined by Wiberg in Chapter 6 of his Ph.D. thesis [Wib96].

We note that **(3.10)** is a special case of **(3.12)**. It applies specifically to TBTs and carries more information about the relationship between the structure of a particular pseudocodeword and its effect on performance of the min-sum algorithm. A similar simplification is not possible for general graphs, since the term "cycles" cannot be applied to general graphs in the same way as it was applied to TBTs.

Chapter 4

Simulation results

As part of the research for this thesis, the min-sum and sum-product algorithms were simulated on a TBT for the (24, 12, 8) Golay code.

The TBT used is the 12-section 16-state trellis that was introduced in [CFV98]. The minimal conventional trellis for this code contains 256 states at its midpoint, which shows that this TBT representation has the minimum possible state-space size.

4.1 Performance of the min-sum algorithm

The performance of the min-sum algorithm on the Golay TBT was measured for SNRs between 0 dB and 6 dB. Figure 4.1 shows the results of these simulations along with a curve showing the union bound on the performance of a maximum likelihood (ML) decoder. The union bound is known to be tight and to provide a good approximation to the performance of a ML decoder for word error rates below 10^{-3}.

Figure 4.1 shows that the min-sum algorithm applied to the TBT of the Golay code asymptotically achieves ML performace.

Since the min-sum algorithm is optimal on the bi-infinite "unwrapped" TBT, its asymptotic performance is limited by the weight and number of minimum distance codewords of the "unwrapped" TBT - i.e. by the weight and number of minimum-effective-weight pseudocodewords of the TBT. The fact that the min-sum algorithm is

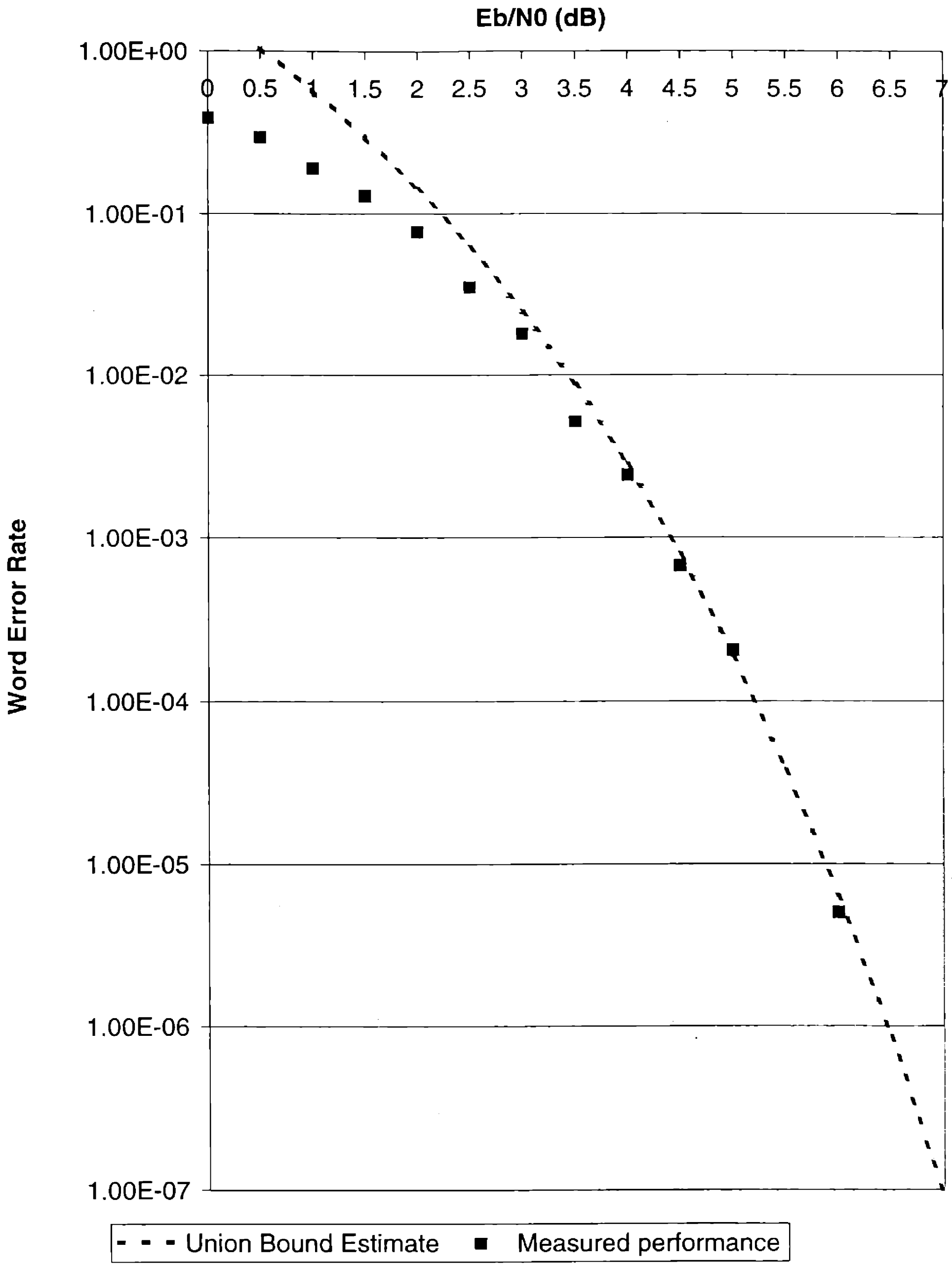

Figure 4.1 Performance of the min-sum algorithm on the TBT of the (24, 12, 8) Golay code.

asymptotically ML on the Golay TBT implies that there are no pseudocodewords of this TBT with effective weight less than 8 (since 8 is the minimum weight of the valid codewords). Moreover, the number of pseudocodewords that are not valid codewords of effective weight 8 must be small compared to the number of weight-8 codewords, because the effect of a substantial number of additional weight-8 error patterns would cause the performance curve of the min-sum algorithm to lie asymptotically somewhat above the union bound. It is most likely, however, that the minimum effective weight of pseudocodewords that are not valid codewords is greater than 8.

The above conjecture could be confirmed analytically by examining the Golay TBT. However, time constraints have not permitted us to implement and execute the exhaustive search algorithm that would be necessary. However, we have examined an extensive number of pseudocodewords observed during the simulations of the min-sum algorithm on the Golay TBT, and we have found that the minimum effective weight observed was 9, 1 greater than the minimum Hamming weight 8 of the code. These results support the conjecture that the minimum effective weight of pseudocodewords that are not valid codewords is greater than 8. In Section 4.2 we present some details of this investigation.

4.2 Analysis of observed errors

The simulation software used to simulate the min-sum algorithm used a stopping rule that looked for convergence of the algorithm. In the absence of convergence (as would happen in the case of an invalid pseudocodeword), it stopped after some maximum number of iterations. The maximum number of iterations was set at 100, which is large enough to ensure convergence of the algorithm to a pseudocodeword.

The output of the algorithm was then a single-cycle section of a pseudocodeword from which the rest of the pseudocodeword had to be extrapolated. The extrapolation was done by trying to find the most likely extension to a 2-cycle pseudocodeword.

The choice of the "most likely" extrapolations was largely determined by the data. Sometimes the data presented two separate pieces of the pseudocodeword in the single-cycle "snapshot." Alternatively, a clear single winner path emerged that did not start and end in the same state, and these provided starting points for completing the pseudocodeword. Also, there were occasions where the winner path and the second best path were so close in their metric that one suspected these to be two pieces of a pseudocodeword. However, the extrapolation always left one with a number of possibilities to consider.

Below we present examples of several 2-cycle pseudocodewords found by this technique, and calculate their effective weights. In presenting the pseudocodewords, we group the sequence bits in pairs. This is done because the Golay TBT of [CFV98] has a section length of 2 bits, so such a representation makes it easier to trace the pseudocodewords. The two cycles of the pseudocodeword are written one under the other to make it easier to calculate the effective weight.

Example 1:

00 00 00 00 11 01 00 01 00 00 10 11

00 00 10 10 01 00 00 11 00 00 00 00

- average Hamming weight per cycle = 6
- $E_1 = 8$ (the number of positions where 0 and 1 overlap)
- $E_2 = 2$ (the number of positions where 1 and 1 overlap)
- $W_{eff} = \frac{(1 \cdot 8 + 2 \cdot 2)^2}{1^2 \cdot 8 + 2^2 \cdot 2} = \frac{144}{16} = 9$

Example 2: A second possibility for the same observed output as Example 1.

00 00 00 00 11 01 00 01 00 00 10 11

00 00 10 01 00 10 11 00 00 00 00 00

- average Hamming weight per cycle = 6
- $E_1 = 12$
- $E_2 = 0$
- $W_{eff} = 12$

Because of the difference in effective weights we conclude that the pseudocodeword shown in Example 1 was much more likely to have occurred then the one shown in Example 2.

Example 3:

00 00 00 00 11 00 00 01 10 10 00 00

00 10 00 10 10 10 10 11 00 00 00 00

- average Hamming weight per cycle = 6
- $E_1 = 8$
- $E_2 = 2$
- $W_{eff} = 9$

Example 4: Another possibility for the same observed output as Example 3.

00 00 00 00 11 00 00 01 01 01 01 00

00 00 10 00 01 01 10 11 00 00 00 00

- average Hamming weight per cycle = 6
- $E_1 = 8$
- $E_2 = 2$
- $W_{eff} = 9$

Example 5:

00 00 10 00 00 00 10 01 10 00 00 11

11 00 10 00 10 00 00 11 00 00 00 00

- average Hamming weight per cycle = 6
- $E_1 = 8$
- $E_2 = 2$
- $W_{eff} = 9$

Example 6: Another possibility for the same observed output as Example 5.

11 10 00 00 00 10 01 10 00 00 11 11

00 10 00 10 00 00 00 10 00 01 01 10

- average Hamming weight per cycle = 8
- $E_1 = 8$
- $E_2 = 4$
- $W_{eff} = 10.67$

We note that the increase in W_{eff} from Example 5 to Example 6 is smaller then the increase in the average Hamming weight per cycle. This is explained by the fact that the four additional 1's increased E_2, while E_1 remained the same.

If the additional 1's contributed to E_1 instead, keeping E_2 constant, W_{eff} would be equal to 12.8, and the increase in W_{eff} would be greater then the increase in the average Hamming weight. Even if the four additional 1's were distributed equally between E_1 and E_2, W_{eff} would be equal to 11.64 and the increase in W_{eff} would still be greater then the increase in the average Hamming weight.

This example is a good illustration of the fact that the structure of the pseudocodeword influences its effect on the performance of the min-sum algorithm.

Example 7:

00 10 10 00 10 00 00 00 10 00 10 00

00 10 00 10 00 00 00 01 10 11 00 10

- average Hamming weight per cycle = 6
- $E_1 = 8$
- $E_2 = 2$
- $W_{eff} = 9$

It is interesting to note that this trellis path of this pseudocodeword never goes through the 0 state.

4.3 Convergence speed

In analyzing the speed of convergence we were primarily concerned with the following metric. If the exact LS procedure were used on a given TBT, or if the code were decoded using a conventional trellis, then the time to execute this procedure would be roughly equal to $2MS^2$, where M is the length of a cycle of the TBT and S is the state space size of the TBT.

It takes an iterative algorithm (min-sum or sum-product) time $2MS$ to go around one cycle (in both directions). Thus, if it takes an algorithm more then an average of S cycles to converge, then its complexity is comparable or greater then that of the LS procedure. However, while the LS procedure is an exact ML decoding algorithm, the min-sum or sum-product algorithms are not exact on the TBT. Therefore, it would not make sense to use the min-sum or sum-product algorithms in this case.

The TBT of the (24,12,8) Golay code has state-space size $S = 16$. Both the min-sum and the sum-product algorithm were found to converge in significantly fewer than 16 iterations around the loop. To test for convergence, our software checked the newly computed normalized vector of probabilities for a given state variable against the previous one. If the difference was found to be below 1×10^{-4} for *all* 16 states, then we stopped the algorithm. The check was performed at every step of the algorithm. Alternatively, the algorithm was stopped after 100 complete iterations. However, we found that this condition occurred extremely rarely.

At low SNR's (around 0 dB), it took the sum-product algorithm an average of about 6 iterations to converge. This number quickly fell as the SNR is increased. At 2 dB, it took the algorithm only about 4 iterations to converge. To achieve word error rates which would be used in practice (10^{-3} and below), SNR's of 4 dB and above were required. At a SNR of 4 dB, only about 2.1 iterations on average were required for the algorithm to converge. At higher SNRs, the number fell below 2 iterations. Similar numbers were observed for the min-sum algorithm.

The results presented above differ somewhat from the numbers given in [AH98]. The authors of [AH98] observed convergence in not much more then one cycle of the

TBT they investigated. However, over the range of SNRs investigated for this thesis (0 through 6 dB), the average number of cycles needed to converge remained well above 1. In the high-SNR limit (with E_b/N_0 set at 100 dB) the algorithm converged after $1\frac{1}{3}$ iterations.

This difference can possibly be explained by a conjecture by Forney [For98] that the convergence time may be related to the memory of the TBT, independent of the cycle length. The memory of the TBT investigated in [AH98] is 1 and the cycle length is large, whereas the memory of the TBT simulated for this thesis is 4 and the cycle length is 12.

Chapter 5

Conclusions and directions for future work

5.1 Summary and conclusions

The research conducted for this thesis concentrated on two main topics: finding an approach for exact decoding of graphs, and investigating lower-complexity sub-optimal iterative approaches.

In searching for an exact decoding method on graphs, we found that the procedure developed by Lauritzen and Spiegelhalter in [LS88] can be applied to TWL graphs. However, this results in increased complexity.

In fact, we demonstrated that constructing a junction tree with **(R)** for a particular code is equivalent to constructing a cycle-free TWL graph for the same code as discussed in Chapter 7 of [Wib96]. This implies that the complexity of the resulting approach is lowerbounded by the "cut-set bound" of [Wib96]; i.e., it has complexity comparable to decoding the minimal conventional trellis of the same code. In practice this implies that the LS procedure will be no less complex then other known exact methods.

We next turned our attention to iterative algorithms on tail-biting trellises (TBTs) as a starting point for a study of how these algorithms behave on general graphs. In particular, we studied the behavior of the min-sum algorithm on TBTs. Here we were able to build on the previous results of [FKV97] and [FKM98] and extend the notion of Hamming weight to pseudocodewords. The result we obtained turned out to be equivalent to the "generalized weight" introduced in Chapter 6 of [Wib96].

We were also able to obtain tight lower and upper bounds for the effective weight of pseudocodewords of TBTs, and showed that for all pseudocodewords that are not valid codewords the effective weight will always be greater then the average Hamming weight per cycle. This implies that the min-sum algorithm can have near ML performance on a TBT.

The min-sum algorithm was simulated on the TBT of the (24, 12, 8) Golay code introduced in [CFV98]. The results of the simulation showed that the min-sum algorithm was asymptotically optimal, in spite of the fact that pseudocodewords of average Hamming weight below the minimum Hamming weight of the code appeared. The effective weight of all observed pseudocodewords turned out to be above the minimum Hamming weight of the code, thus confirming that the effective weight, rather then the average Hamming weight per cycle, is the correct measure of probability of error to a pseudocodeword on an AWGN channel.

The simulation also confirmed that on average the complexity (i.e., the number of operations performed) of the min-sum algorithm is less than that of exact algorithms (e.g. VA decoding of a conventional trellis). In Chapter 4 we observed that at 10^{-3} word error rate the min-sum algorithm needs only about two iterations to converge, which is about 8 times fewer operations than an exact approach. However, as the graph in Figure 4.1 shows, the min-sum algorithm at this word error rate achieves very near ML performance. Thus, we have shown experimentally that for some codes suboptimal iterative decoding algorithms can result in near-ML performance while significantly reducing the computational complexity of decoding.

5.2 Directions for future work

The thesis leaves open several questions. The results of Chapter 3 imply that in constructing TBTs (or more general TWL graphs) for codes, one should maximize the minimum effective weight of pseudocodewords. In particular, one would like to obtain a graph such that the minimum effective weight of the graph is not less than the minimum Hamming weight of the code. It is of interest to investigate the properties needed to meet this objective.

Also, we note that both in this thesis and in [Wib96], the effective weight has been defined only for binary codes, assuming binary antipodal signaling for transmission. With more general modulation schemes, the Euclidean distance in signal space governs the probability of error. It would be desirable to generalize the notion of Euclidean distance to an "effective Euclidean distance" of pseudocodewords.

Bibliography

[Agr97] D. Agrawal. Personal communication. 1997.

[AH98] J. B. Anderson and S. M. Hladik. Tail-biting MAP decoders. *IEEE Journal on Selected Areas in Communication*, 16:297–302, 1998.

[BCJR74] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, 20:284–287, 1974.

[BGT93] C. Berrou, A. Glavieux, and P. Thitimajshima. Near-Shannon-limit error-correcting coding and decoding: Turbo codes. In *Proceedings of IEEE International Conference on Communications (ICC)*, pages 1064–1070, Geneva, Switzerland, 1993.

[BM96] S. Benedetto and G. Montorsi. Unveiling turbo codes: Some results on parallel concatenated coding schemes. *IEEE Transactions on Information Theory*, IT-42:409–428, 1996.

[CFV98] A. R. Calderbank, G. D. Forney, Jr., and A. Vardy. Minimal tailbiting trellises: The Golay code and more. *IEEE Transactions on Information Theory*, 1998. To appear.

[FKM98] G. D. Forney, Jr., F. R. Kschischang, and B. Marcus. Iterative decoding of tail-biting trellises. IEEE Information Theory Workshop, San Diego, February 1998.

[FKV97] B. Frey, R. Kötter, and A. Vardy. Skewness and pseudocodewords in iterative decoding. Submitted to ISIT 98, December 1997.

[For97] G. D. Forney, Jr. On iterative decoding and the two-way algorithm. In *Proceedings of International Symposium on Turbo Codes*, pages 12–25, Brest, France, 1997.

[For98] G. D. Forney, Jr. Personal communication. 1998.

[Gal62] R. G. Gallager. Low-density-parity-check codes. *IRE Transactions on Information Theory*, IT-8:21–28, 1962.

[Gal63] R. G. Gallager. *Low-Density-Parity-Check Codes.* MIT Press, Cambridge, MA, 1963.

[Jen96] F. V. Jensen. *An Introduction to Bayesian Networks.* University College London Press, London, England, 1996.

[Jor] M. I. Jordan. Unpublished course notes for MIT course 9.390, Computational Laboratory in Cognitive Sciences, Spring 1997.

[KF98] F. R. Kschischang and B. J. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *IEEE Journal on Selected Areas in Communication*, 16:219–230, 1998.

[Lau96] S. L. Lauritzen. *Graphical Models.* Clarendon Press, Oxford, Great Britain, 1996.

[LS88] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems. *Journal of the Royal Statistical Society, Series B*, 50:157–224, 1988.

[MMC98] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng. Turbo decoding as an instance of Pearl's 'belief propagation' algorithm. *IEEE Journal on Selected Areas in Communication*, 16:140–152, 1998.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan-Kaufmann, San Francisco, CA, 1988.

[Tan81] R. M. Tanner. A recursive approach to low-complexity codes. *IEEE Transactions on Information Theory*, IT-27:513–526, 1981.

[Wei97] Y. Weiss. Belief propagation and revision in networks with loops. Technical report, Artificial Intelligence Laboratory, MIT, Memo No. 1616, 1997.

[Wib96] N. Wiberg. *Codes and Decoding on General Graphs.* PhD thesis, University of Linköping, Linköping, Sweden, 1996.

[Wic95] S. B. Wicker. *Error Control Systems for Digital Communication and Storage.* Prentice-Hall, Englewood Cliffs, NJ, 1995.

[WLK95] N. Wiberg, H.-A. Loeliger, and R. Kötter. Codes and iterative decoding on general graphs. *European Transactions on Telecommunication*, 6:513–526, 1995.