Chapter 1

INTRODUCTION

"The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point."

—Claude Shannon

This chapter presents a few simple examples to illustrate the principles involved in *soft iterative decoding* (SID) in Section 1. A brief summary of digital communications and potential applications is given in Sections 2 and 3, followed by an overview of the chapters in this book in Section 4. A guide to notation and symbols is provided in Section 5.

1. SOFT ITERATIVE DECODING

"Soft" decoding refers to the use of probabilistic information as input to a decoding algorithm, as opposed to "hard" decoding, where the input is a sequence of symbols (e.g., bits). Section 1.1 gives some problems related to coin probabilities, illustrating the idea of soft decoding. In Section 1.2, the counting of soldiers illustrates the use of iterative decoding to perform a computation. Section 1.3 describes the message-passing algorithm (MPA), which is a soft iterative decoding technique.

1.1 COIN PROBABILITIES

We consider a puzzle related to coin probabilities: Suppose there are three coins, which are known to be all head or all tails. In addition, suppose that the first coin is known to be a head with probability $\frac{1}{3}$, the second coin is (independently) known to be a head with probability $\frac{1}{3}$, and the third coin has a uniform distribution (with probability $\frac{1}{2}$). Given this information, what is the probability that the third coin is a head? (Hint: The answer is neither $\frac{1}{3}$ nor $\frac{1}{9}$.)

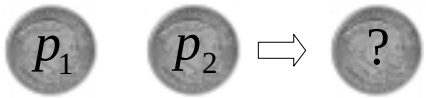

Figure 1.1. Evaluating probabilities on coins

Let us describe this problem more precisely. Let x_i denote the i-th coin, for $i = 1, 2, 3$, with the value $x_i = 1$ representing heads and the value $x_i = 0$ representing tails. The first coin is known to be a head with probability $p_1 = P(x_1 = 1)$, and the second coin is known to be a head with probability $p_2 = P(x_2 = 1)$. Nothing is known about the third coin, so it is assumed to have a uniform distribution, $p_3 = P(x_3 = 1) = \frac{1}{2}$. These probabilities p_1, p_2, and p_3 are known as the *a priori* (or "prior", for short) probabilities, and can come from an observation or external source.

Additional structure on the configuration of the coins can be described by a constraint set $S = \{(x_1, x_2, x_3)\}$ of valid configurations. In the case where the coins are all tails or all heads, the constraint set is

$$S = \{(0,0,0), (1,1,1)\}.$$

Using this information, the problem is to find the probability that $x_3 = 1$, given that the three coins (x_1, x_2, x_3) satisfy the constraint S. This conditional probability is known as the *a posteriori* (or "posterior", for short) probability, and can be written as

$$p_3^{\text{post}} = P(x_3 = 1 \mid (x_1, x_2, x_3) \in S).$$

(Prior and posterior probabilities are discussed in more detail in Chapter 2, Section 1.)

Recall the rule for conditional probabilities (ref. [Leo94]):

$$P(A \mid B) = \frac{P(A, B)}{P(B)}$$

In this case, let A represent the event that the third coin is a head ($x_3 = 1$), and let B represent the event that all three coins satisfy the constraint ($(x_1, x_2, x_3) \in S$). Then the posterior probability is

$$p_3^{\text{post}} = \frac{\sum_{\substack{(x_1,x_2,x_3)\in S \\ x_3=1}} P(x_1, x_2, x_3)}{\sum_{(x_1,x_2,x_3)\in S} P(x_1, x_2, x_3)}. \tag{1.1}$$

The summation is over all configurations in S such that $x_3 = 1$, normalized by summing over the probabilities of all configurations in S. It is assumed that the prior probabilities are independent, so that the joint probability can be factored as follows:

$$P(x_1, x_2, x_3) = P(x_1) P(x_2) P(x_3)$$

- **Example 1 (equality).** When the coins are either all heads or all tails, the set of valid configurations for the three coins is denoted $S = \{(0,0,0), (1,1,1)\}$. Using (1.1), the probability of the third coin being a head is

$$p_3^{\text{post}} = \frac{p_1 p_2 p_3}{p_1 p_2 p_3 + (1-p_1)(1-p_2)(1-p_3)}.$$

  Then suppose that the prior probability that the first coin is a head ($x_1 = 1$) is $p_1 = \frac{1}{3}$, and similarly for the second coin, $p_2 = \frac{1}{3}$. Since nothing is known about the third coin, $p_3 = \frac{1}{2}$ is assumed. Then the posterior probability is

$$\frac{(1/3)^2}{(1/3)^2 + (2/3)^2} = 1/5,$$

  so that having two probabilities of $1/3$ decreases the certainty that the third coin is a head to $1/5$, answering the question posed in the first paragraph.

- **Example 2 (parity-check).** Next, suppose the structure is that there are an even number of heads amongst the three coins. In this case, the set of valid configurations is

$$S = \{(0,0,0), (1,1,0), (1,0,1), (0,1,1)\}.$$

  If the first two coins have prior probabilities p_1 and p_2, then using (1.1), the posterior probability of the third coin can be deduced to be

$$p_3^{\text{post}} = p_1(1-p_2) + (1-p_1) p_2.$$

  In other words, the probability that the third coin comes up heads is given by the probability that exactly one of the other two coins is a head. If $p_1 = p_2 = \frac{1}{3}$, then nothing has been learned about the third coin, as $p_3^{\text{post}} = \frac{1}{2}$.

- **Example 3 (constraint).** In general, the constraint set can be an arbitrary restriction on the allowable patterns. Suppose that the constraint permits exactly one head amongst the three coins, so that the constraint set is

$$S = \{(1,0,0), (0,1,0), (0,0,1)\}.$$

  Given the prior probabilities p_1 and p_2 for the first and second coin, the posterior probability for the third coin can be found as follows:

$$\begin{aligned} p_3^{\text{post}} &= \frac{(1-p_1)(1-p_2)}{p_1(1-p_2) + (1-p_1) p_2 + (1-p_1)(1-p_2)} \\ &= \frac{(1-p_1)(1-p_2)}{1-p_1 p_2} \end{aligned}$$

  In this case, even when the first two coins have uniform probabilities $p_1 = p_2 = \frac{1}{2}$, the knowledge that the coins satisfy the constraint S gives a posterior probability of $p_3^{\text{post}} = \frac{1}{3}$ for the third coin to be a head.

The first two examples, "equality" and "parity-check," correspond to the equality node and parity-check node discussed in Chapter 2. The third example, "constraint," gives an example of a non-linear constraint similar to the constrained codes discussed in Chapters 5 and 6. The message-passing algorithm is based on simple probabilistic calculations such as these, which are combined to solve complicated decoding problems.

1.2 COUNTING SOLDIERS

An example due to Pearl [Pea88] (also presented in [Mac01]) serves to illustrate how the passing of messages in a graph can solve a problem as a distributed computation. First, suppose there is a troop of soldiers standing in a row, as pictured in Figure 1.2. Each soldier can only communicate with the neighboring soldiers directly ahead and behind. The problem is to count the total number of soldiers and distribute this information to every soldier.

An interesting solution is given by the following rule: *Whenever you receive a number from one neighbor, add 1 and pass this number to the other neighbor.* The algorithm starts with the soldiers standing at the ends of the line, who each pass a 1 to their immediate neighbors. The numbers are incremented by 1 each time they get passed along. The interpretation of the messages can be as follows: If a soldier passes a message n to another soldier, it means, "I believe that there are n soldiers, including myself, upstream of this message." When all messages have been passed, each soldier sums the incoming numbers from the neighbors, and then adds 1 more to obtain the total number of soldiers.

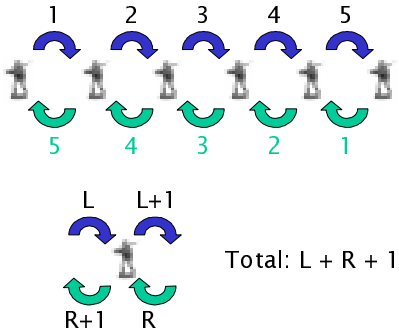

Figure 1.2. Counting soldiers using message-passing

A graph can be used to represent the soldiers, as shown in Figure 1.3. The nodes represent the soldiers, and the edges represent the communication links between the soldiers. The relevant rule for message-passing for graphs in this case is as follows: *If a node has $K+1$ edges, sum the incoming messages along each subset of K edges, and pass the sum along the remaining edge.* In other words, if a node has $K+1$ edges, then any time it receives incoming messages along K edges, it sums those numbers and passes the result out along the remaining edge. If a node receives less than K messages, then it does nothing, and waits for more messages to arrive. (Once a message is passed along an edge, it is assumed to stay there until it is updated.) For the nodes in Figure 1.3, there is one vertical edge that is used for inputting the initial count of 1, and one or two horizontal edges representing the connections to neighboring soldiers.

Using these rules, the order of the message-passing in the graph is shown in Figure 1.4. First there is an edge to each node with an incoming message of 1, representing each soldier's initial count that there is 1 soldier—namely himself or herself. Next, since the two nodes at the ends have only 2 edges, the one incoming message is sufficient to trigger an outgoing message to the neighboring soldier. Subsequently, in both the

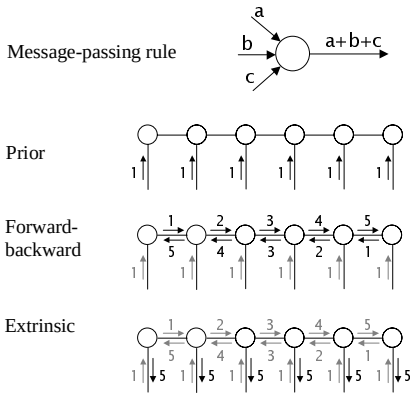

Figure 1.3. Graphical representation for soldier-counting

forward and backward directions, each message passed to a node triggers that node to send a message in the same direction to the next node, with the value incrementing by 1 each time. Finally, when a node receives a message from both the left and right sides, it sums the numbers together to give an outgoing message along the bottom edge. (For the case of the two soldiers on the end, which only have two edges, any message they receive along one edge is directly passed along the other.) This outgoing message is called the "extrinsic" message, and represents the additional knowledge that the soldier has learned from the message-passing. (The labels in Figure 1.3 of "intrinsic," "forward-backward," and "extrinsic" refer to the message-passing algorithm in Chapter 2 and the forward-backward algorithm in Chapter 3.) In other words, the value of the output message, which is 5 in this case, says that the message-passing has counted 5 additional soldiers in the graph. Putting this together with the initial message of 1 gives the total number of soldiers correctly at 6.

This message-passing rule works perfectly whenever the graph contains no cycles (i.e., paths that begin and end in the same node), as

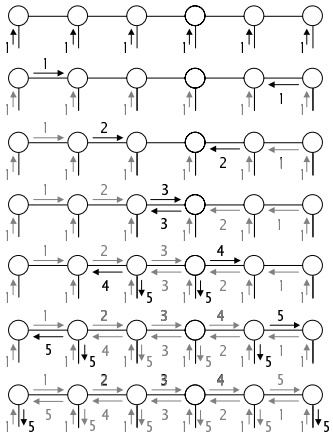

Figure 1.4. The order in which messages are passed in the graph

shown in Figure 1.5, allowing for the computation of the total number of soldiers in a distributed manner. The situation of a soldier with 3 neighboring soldiers is illustrated in more detail in Figure 1.6, where the soldier is represented by a node that has 4 edges (3 for the neighbors, and 1 for the soldier's own count). Then the output to a neighboring soldier is equal to the sum of the inputs from the other two neighbors plus 1.

On the other hand, this method of soldier-counting can fail badly when there is a cycle in the graph. Suppose that a group of soldiers stands in a circle, as shown in Figure 1.7. Using the message-passing rule to count the total number of soldiers results in the numbers incrementing to infinity!

1.3 MESSAGE-PASSING ON GRAPHS

The key idea of the message-passing algorithm, as illustrated by the soldier-counting example, is to solve a problem using simple computation units that exchange messages. We give a brief description of a graph

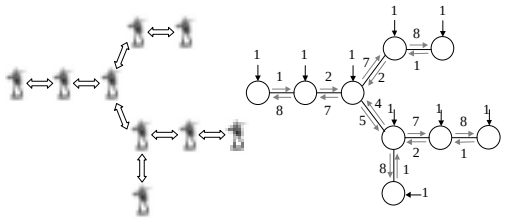

Figure 1.5. Message-passing works for counting soldiers in a tree formation

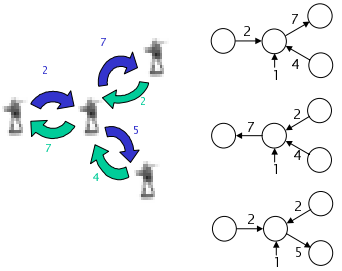

Figure 1.6. A soldier with three neighbors

structure (Forney's normal graphs [For01]) that can be used to perform message-passing. As shown in Figure 1.8, the graph is an undirected graph, in which a variable is associated with each edge, and each node represents a *local constraint* on the variables associated with the edges connected to that node. Messages corresponding to the probability distributions for the variable are passed in both directions along all edges of the undirected graph.

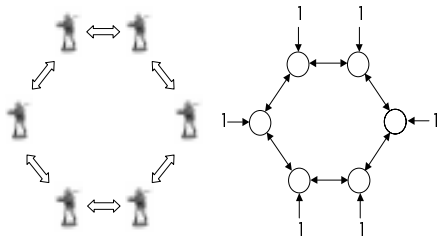

Figure 1.7. Message-passing fails for soldiers in a circle formation

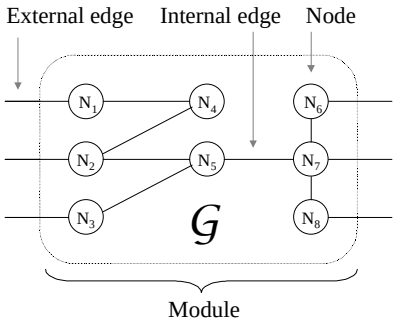

Figure 1.8. Graph consisting of nodes and edges

For the graph, there may also be edges that are only connected to one node. These are the *external edges* (also known as "half-edges") as opposed to the *internal edges* that connect two nodes. These external edges are used for passing probabilistic information in and out of this structure. Since both the input and output are message vectors corresponding to probabilities, this decoding process is often called "soft-in, soft-out" (SISO). The overall decoding system consists of different nodes that exchange probabilistic information. In addition, it is possible to group these nodes and edges into *modules*, which can be connected together to form larger decoding systems. Nodes, modules and the message-passing algorithm are discussed in detail in Chapter 2.

2. DIGITAL COMMUNICATIONS

We describe a framework for digital communications, starting first with the channel and then moving on to coding and modulation.

2.1 CHANNEL

Digital communications is the art of getting bits reliably from one place to another (or, in the case of digital storage, from one time to another) across an unreliable channel. Let the sequence x represent a collection of signals x_i; this may be a finite block $x = (x_1, x_2, ..., x_N)$ or a bi-infinite sequence $x = (..., x_{i-1}, x_i, x_{i+1}, ...)$. As shown in Figure 1.9, the channel may introduce noise and distortion to the transmitted sequence x, which is received as the sequence y. The goal of the *channel detector* is to use the received sequence y and knowledge of the channel characteristics to produce a replica $\widehat{x}$ of the original sequence x. *Maximum-likelihood sequence detection (MLSD)* [For72] consists of choosing the sequence that maximizes the probability that the received vector is y if the transmitted sequence x was sent.

$$\widehat{x}^{ML} = \arg\max_{x} \Pr(y \mid x).$$

In our terminology, the term "decoder" is used generically to describe any process that involves the recovery of a data sequence that has been "encoded," where encoding is broadly defined to include the transformation introduced by the channel. As a result, the term "*channel decoder*" will be used interchangably with "channel detector" to describe the module that performs computations to obtain information on transmitted sequence.

Some key examples of channels are the *binary symmetric channel* (BSC), the *binary erasure channel* (BEC), and the *additive white Gaussian noise* (AWGN) channel, as shown in Figure 1.10. For the BSC and BEC, the transmitted signals $\{x_i\}$ are binary, and the received signals $\{y_i\}$ are discrete-valued as well. On the other hand, for the AWGN channel, the signals are real-valued (or complex-valued), and the received signal y_i is given by

$$y_i = x_i + n_i,$$

Chapter 2

MESSAGE-PASSING ALGORITHM

E pluribus unum (out of many, one)

from the Great Seal of the United States of America

This chapter introduces a framework for understanding probabilistic decoding, and applies it to the decoding of low-density parity check (LDPC) codes. Section 1. sets up the preliminaries for soft iterative decoding. Section 2. explains the message-passing algorithm using normal graphs. Section 3. presents some basic nodes and modules. Section 4. introduces LPDC codes and discusses their decoding algorithm.

1. PRELIMINARIES

1.1 PROBABILITIES

This section reviews some basic definitions and concepts in probability. [Leo94]

1.1.1 BASIC CONCEPTS

For an event E, we can assign a probability $P(E)$, which is a real number between 0 and 1, inclusive. It is useful to consider think of E as a set of outcomes, so that $P(E)$ is the probability of choosing an outcome from that set. For the null set $\emptyset$, and the sample space $\mathcal{A}$ consisting of all outcomes, we have $P(\emptyset) = 0$ and $P(\mathcal{A}) = 1$. Suppose that a set of events $\{F_k\}$, for $k = 1, 2, ..., K$, forms a partition of the complete set of events, so that two events are not simultaneously true, and the set covers the complete space of events. In set notation, $F_k \cap F_{k'} = \emptyset$ if $k \neq k'$, and $\bigcup_k F_k = \mathcal{A}$, where $P(\mathcal{A}) = 1$. Then the *total probability* is

equal to 1.

$$\sum_{k=1}^{K} P(F_k) = 1. \tag{2.1}$$

For two events E and F, the joint probability $P(E,F)$ is the probability that both events are true. In set notation, this corresponds to the probability of choosing an outcome from the intersection $E \cap F$.[1] For any event E, the probability $P(E)$ can be decomposed as the sum of the joint probabilities of E and F_k.

$$P(E) = \sum_{i=1}^{K} P(F_k, E). \tag{2.2}$$

Two events E and F are said to be *independent* exactly when the joint probability factors into the product of the individual probabilities:

$$P(E,F) = P(E)\,P(F) \tag{2.3}$$

For two events E and F, the *conditional probability* $P(F \mid E)$ is the probability that an event F is true given the event E is true, which can be expressed as follows,

$$P(F \mid E) = \frac{P(E,F)}{P(E)} \tag{2.4}$$

for $P(E) > 0$. In the case that the events E and F are independent, it can be seen from (2.3) and (2.4) that

$$P(F \mid E) = P(F) \quad \text{(if } E, F \text{ independent)}, \tag{2.5}$$

so that conditioning does not change the probability if the events are independent. Applying the conditional probability expression once again to the numerator of (2.4) yields *Bayes' theorem* (also known as Bayes' rule):

$$P(F \mid E) = \frac{P(E \mid F)\,P(F)}{P(E)} \tag{2.6}$$

In addition, for probabilities conditioned on two events, a similar result as (2.4) holds,

$$P(E, G \mid F) = P(E \mid F, G)\,P(G \mid F), \tag{2.7}$$

[1] This joint probability satisifes the following upper and lower bounds:

$$P(E) + P(F) - 1 \leq P(E,F) \leq \min\{P(E), P(F)\}$$

which is known as the *chain rule* for conditional probabilities. A special case of (2.7) occurs when

$$P(E, G \mid F) = P(E \mid F)\, P(G \mid F), \tag{2.8}$$

in which case the three events E, F and G (in that order) are said to satisfy the *Markov property*. These events then form a *Markov chain*, which is denoted by $E \leftrightarrow F \leftrightarrow G$. Two conditions for a Markov chain that are equivalent to (2.8) are

$$\begin{aligned} P(E \mid F, G) &= P(E \mid F) \\ P(G \mid E, F) &= P(G \mid F)\,. \end{aligned}$$

In this Markov chain, conditioning by the event F makes the other events E and G independent, as in (2.3).

Concepts such as independence, Bayes' theorem and Markov chains provide the basic tools for the definition of the decoding algorithms in Section 2.

1.1.2 ALPHABETS AND PROBABILITY VECTORS

An *alphabet* A represents a set of values that a variable can take. The alphabet can be a discrete set, such as $A = \{0, 1\}$, or a continuous set, such as $A = \mathbb{R}$. (For this discussion, we restrict our attention to discrete sets.) The alphabet may have some additional structure, such as being a group under addition (e.g., the additive group of m elements, $\mathbb{Z}_m$), or a field (e.g., the Galois field $\mathbb{F}_q = GF(q)$).

For an event E, it is possible to assign a probability $P(E)$, which is a real number between 0 and 1, inclusive. Then for a random variable x belonging to an alphabet A, it is possible to consider the event $\{x = a\}$ that x equals some particular element $a \in A$. The probability of this event is represented by $P(x = a)$. As a function of $a \in A$, this probability $P(x = a)$ gives a *probability mass function* for the variable x.

Since the set of events $\{x = a\}$, for all $a \in A$, forms a partition of the complete set of events, the probabilities $P(x = a)$ over all values in the alphabet A sum to 1.

$$\sum_{a \in A} P(x = a) = 1$$

These probabilities can be represented by a vector as well, which is called a vector of probabilities, or *probability vector*,

$$\overrightarrow{P}(x) = \begin{bmatrix} P(x = a_1) \\ P(x = a_2) \\ \vdots \\ P\left(x = a_{|A|}\right) \end{bmatrix}, \tag{2.9}$$

where the alphabet A consists of $\left\{a_1, a_2, ..., a_{|A|}\right\}$. The notation $\overrightarrow{P}(x)$ then represents the entire vector of probabilities shown in (2.9). Note that the notation $P(x)$, i.e., without the arrow, is defined in Section 2.1.3, to represent $P(x = a)$ for some value $a \in A$, which can be determined from the context.

On the other hand, an arbitrary vector of non-negative real numbers (that do not necessarily sum to 1) is called a *message vector* and is denoted by the symbol $\overrightarrow{\mu}$. If the variable x belongs to the alphabet A, then the message vector would have length $|A|$,

$$\overrightarrow{\mu}(x) = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_{|A|} \end{bmatrix},$$

where $m_k \geq 0$, and it is assumed that the entries of the message vector are not all simultaneously zero. The entries of the vector can also be written as $\mu(x = a_k) = m_k$, in analogy to the probability notation $P(x = a_k)$.

It is possible to set up an equivalence relation ($\equiv$) on message vectors such that each message vector is equivalent to a probability vector by normalizing it by the sum of all elements.

$$\begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_{|A|} \end{bmatrix} \equiv \frac{1}{\sum_{k=1}^{|A|} m_k} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_{|A|} \end{bmatrix}$$

As a result of this equivalence relation, there are $|A| - 1$ degrees of freedom in the message vector.[2]

[2]Note that since the message vector $\begin{bmatrix} m_1 & m_2 & \cdots & m_{|A|} \end{bmatrix}^T$, satisfies an equivalence relation such that multiplication by any positive constant $c > 0$ yields an equivalent message

We will also consider the set defined by the product of multiple alphabets $A_{x_0} \times A_{x_1} \times \cdots \times A_{x_K}$, corresponding to the all possible values of the variables, $x_0, x_1, ..., x_K$. Each element

$$(\xi_0, \xi_1, ..., \xi_K) \in A_{x_0} \times A_{x_1} \times \cdots \times A_{x_K}$$

of this set is called a *configuration*, where each value ξ_i is selected from the appropriate alphabet A_i, and corresponds to a possible value for the variable x_i.

1.1.3 INTRINSIC, EXTRINSIC AND POSTERIOR

For a variable x, there are several types of probabilities that can be associated with the variable. For the event $\{x = a_k\}$, suppose that E is an event whose effect on the variable x is under question. The *prior* (or *a priori*) probability refers to the probability $P(x = a_k)$ that the variable x takes the value a_i. In the iterative algorithms that we consider, however, the prior probability that is used for a variable x may depend on the choice of the event E, so that it is more appropriate to speak of the prior probability for x *with respect to* E. To avoid confusion with the true prior probability, we substitute the term "*intrinsic*" for "prior" to describe this probability. The *intrinsic probability for* x *with respect to* E is denoted by

$$P_E^{\text{int}}(x = a) = P(x = a)\,.$$

On the other hand, the *posterior* (or *a posteriori*) probability is the conditional probability based on knowledge of the event E. The *posterior probability for* x *with respect to* E is denoted

$$P_E^{\text{post}}(x = a) = P(x = a \mid E)\,.$$

The intrinsic and posterior probabilities represent the probability *before* and *after* taking into account the event E.

The posterior probability can be written using Bayes' theorem (2.6) as

$$\overbrace{P(x = a \mid E)}^{\text{posterior}} = \frac{1}{P(E)} \overbrace{P(E \mid x = a)}^{\text{extrinsic}} \overbrace{P(x = a)}^{\text{intrinsic}}. \tag{2.10}$$

vector, it can be interpreted as a point in the projective space $\mathbb{P}^{|A|}(\mathbb{R})$. In particular, the set of message vectors correspond to the points in projective space such that all elements of the vector have the same sign (or are zero). These message vectors occupy $1/2^{|A|}$ of volume of the projective space. In the case of $|A| = 2$, the points in the projective plane $\mathbb{P}^2(\mathbb{R})$ correspond to lines through the origin, and the message vectors correspond to lines $y = mx$ with slope $m \in [0, \infty]$.

The term $P(x = a)$ that appears in the right-hand side of (2.10) is the intrinsic probability. The complementary term $P(E \mid x = a)$ corresponds to the "extrinsic" probability, which is a probability that describes the new information for x that has been obtained from the event E. The *extrinsic probability for* x *with respect to* E is defined by

$$P_E^{\text{ext}}(x = a) = c'_x P(E \mid x = a)\,, \tag{2.11}$$

where the normalization constant

$$c'_x = \frac{1}{\sum_{a \in A} P(E \mid x = a)}$$

is necessary so that the entries in the extrinsic probability vector $P_E^{\text{ext}}(x)$ sum to 1 (i.e., $\sum_{a \in A} P_E^{\text{ext}}(x = a) = 1$).

Then the relationship between the intrinsic, extrinsic and posterior probabilities in (2.10) can be rewritten as

$$P_E^{\text{post}}(x = a) = \widetilde{c}_x P_E^{\text{int}}(x = a)\, P_E^{\text{ext}}(x = a)\,, \tag{2.12}$$

where $\widetilde{c}_x$ is a normalization constant that takes the form

$$\widetilde{c}_x = \left(\sum_{a \in A} P_E^{\text{int}}(x = a)\, P_E^{\text{ext}}(x = a) \right)^{-1}.$$

This constant is necessary in order to make the posterior probabilities sum to 1 (i.e., $\sum_{a \in A} P_E^{\text{post}}(x = a) = 1$). (Comparing (2.10), (2.11) and (2.12), it can be seen that the constants are related by $\widetilde{c}_x = (c'_x P(E))^{-1}$).

In terms of probability vectors, the posterior probability vector is proportional to the message vector that is generated from the pointwise multiplication of these two vectors,[3]

$$\begin{aligned} \overrightarrow{P}_E^{\text{post}}(x) &= \widetilde{c}_x \left(\overrightarrow{P}_E^{\text{int}}(x) \odot \overrightarrow{P}_E^{\text{ext}}(x) \right) \\ &= \widetilde{c}_x \begin{bmatrix} P_E^{\text{int}}(x = a_1)\, P_E^{\text{ext}}(x = a_1) \\ P_E^{\text{int}}(x = a_2)\, P_E^{\text{ext}}(x = a_2) \\ \vdots \\ P_E^{\text{int}}\left(x = a_{|A|}\right) P_E^{\text{ext}}\left(x = a_{|A|}\right) \end{bmatrix} \end{aligned}$$

[3] It should be noted some of these quantities may not be well-defined if there are zeros. For example, if for some k, $P_E^{\text{int}}(x = a_k)$ and $P_E^{\text{post}}(x = a_k)$ are both $\mathbf{0}$, then the extrinsic probability $P_E^{\text{ext}}(x = a_k)$ is not well-defined. In practical implementations, this means that the probabilities are usually bounded away from 0 and 1 (e.g., by ϵ and $1 - \epsilon$, respectively).

where the elements of the alphabet are $\{a_1, a_2, ..., a_{|A|}\}$ and $\odot$ represents the pointwise multiplication of the two vectors.

To recap, the terms "intrinsic," "extrinsic," and "posterior" describe probabilities for a variable in relation to an event. (In the course of the decoding algorithm, the estimated probabilities may differ from the true probabilities, so that it may be more accurate to use terms such as *pseudo-intrinsic, pseudo-extrinsic* and *pseudo-posterior* to indicate that these are not the exact values. For convenience, however, we will continue to use the terms *instrinsic, extrinsic* and *posterior*, with the understanding that these may represent working estimates.) For clarity, the intrinsic, extrinsic or posterior probabilities for x should be labeled "with respect to E" (which can be abbreviated as "w.r.t. E"), and denoted by $P_E^{\text{int}}(x=a)$, $P_E^{\text{ext}}(x=a)$, and $P_E^{\text{post}}(x=a)$. This explicit reference to the event E may be omitted when it is understood from context.

1.1.4 BINARY VARIABLES

For the case of binary variables, which take on values from a binary alphabet $A=\{0,1\}$, there are several equivalent representations of the probability:

1. The *probability* $p = P(x=1)$, which lies in the range $[0,1]$.
2. The *probability vector* $\overrightarrow{P}(x)$, which consists of two probabilities that sum to 1.
$$\overrightarrow{P}(x) = \begin{bmatrix} P(x=0) \\ P(x=1) \end{bmatrix} = \begin{bmatrix} 1-p \\ p \end{bmatrix}$$
3. The *log-likelihood ratio*,
$$\text{LLR}(p) = \log\frac{P(x=1)}{P(x=0)} = \log\frac{p}{1-p},$$
which is a real number.[4]
4. The *soft bit* $\chi(p) = 2p-1$, which lies in the range $[-1,1]$.

By convention, the log-likelihood ratio uses the natural logarithm. The relationship between the soft bit (ref. [HOP96]) and the log-likelihood

[4] It is possible to choose the opposite convention $\log\frac{P(x=0)}{P(x=1)}$ for the definition of the log-likelihood ratio, which turns out to give simpler expressions in the decoding of low-density parity-check codes. We stick with the convention $\log\frac{P(x=1)}{P(x=0)}$ so that a positive LLR corresponds to $x=1$.

ratio can then be expressed using the hyperbolic tangent function

$$\chi(p) = 2p-1 = \tanh\left(\frac{1}{2}\text{LLR}(p)\right), \tag{2.13}$$

where $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. For a binary variable, the log-likelihood ratios corresponding to the intrinsic and extrinsic probabilities sum to give the LLR for the posterior probability,

$$\text{LLR}_E^{\text{post}}(x) = \text{LLR}_E^{\text{int}}(x) + \text{LLR}_E^{\text{ext}}(x), \tag{2.14}$$

where the subscript E indicates that it is with respect to an event E. This can be derived from (2.12), which gives

$$P_E^{\text{post}}(x=1) = \frac{pe}{pe + (1-p)(1-e)},$$

where $p = P_E^{\text{int}}(x=1)$, and $e = P_E^{\text{ext}}(x=1)$. In this log-likelihood representation, it can clearly be seen that the extrinsic information reflects the incremental gain in knowledge of the posterior over the intrinsic information. Note that if the posterior and intrinsic probabilities are equal, then $\text{LLR}_E^{\text{ext}}(x) = 0$.

1.2 NORMAL GRAPH

A *normal graph* $\mathcal{G}$ is an undirected graph, consisting of nodes and edges, as introduced by Forney in [For01]. Each node can be connected to any number of edges, and the edges are connected to one or two nodes. The presence of edges that are connected to only one node is a special property of normal graphs. An edge connected to two nodes is an *internal edge*, while an edge connected to only one node is an *external edge*. (These external edges are also known as "half-edges," "leaf-edges" or "dongles.") To each edge is associated a variable x_i that takes values over a finite, discrete alphabet denoted by A_{x_i}. When referring to the combination of the edge and its associated variable, we use the compound term "*edge-variable*." The variables associated with the external edges are known as the "symbol variables," and represent the coupling of the system with its environment, while the internal edge-variables are known as the "state variables," since they represent internal states of the system.

To each node N is associated a *local constraint*, expressing a relationship amongst the variables on the edges connected to that node. As shown in the top part of Figure 2.1, for a single node N in a graph $\mathcal{G}$, which has $K+1$ edges that are associated with the variables $x_0, x_1, ...,$

	Association	Purpose
External edge	Symbol variable	Input/Output
Internal edge	State variable	Message-passing
Node	Local constraint	Computation

Table 2.1. Components of a normal graph

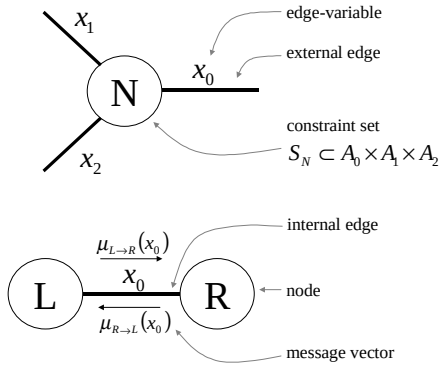

Figure 2.1. Examples of normal graphs

x_K, the set of all combinations of these variables that satisfy the local constraint is dictated by the *constraint set* S_N.

$$S_N \subset A_{x_0} \times A_{x_1} \times \cdots \times A_{x_K}$$

This constraint set is a subset of the direct product of the corresponding alphabets for each of the variables. The role of the edges and nodes in a normal graph is summarized in the Table 2.1.

It will be seen in Section 2.1 that an extrinsic probability $P_N^{\text{ext}}(x_0)$ for the edge-variable x_0 can be calculated at node N based on the intrinsic probabilities $P_N^{\text{int}}(x_i)$ for the other edge-variables connected to node N

as follows,

$$\begin{aligned} P_N^{\text{ext}}(x_0 = \xi_0) &= c_0' P(N \mid x_0 = \xi_0) \\ &= c_0' \sum_{\substack{\xi_1,\xi_2,\ldots,\xi_k \\ (\xi_0,\xi_1,\xi_2,\ldots,\xi_K)\in S_N}} \prod_{i=1}^{K} P_N^{\text{int}}(x_i = \xi_i) \end{aligned}$$

where c_0' is a normalization constant, and the summation is over all combinations of the variables $\xi_1, \xi_2, ..., \xi_k$ (where $\xi_i \in A_i$) such that the configurations $(\xi_0, \xi_1, \xi_2, ..., \xi_K)$ satisfies the constraint set S_N.

Along each edge of the graph are passed message vectors in both directions. For an edge-variable x_0 between nodes L and R, the message vector going from L to R is denoted by $\overrightarrow{\mu}_{L\to R}(x_0)$, while the message vector from R to L is denoted by $\overrightarrow{\mu}_{R\to L}(x_0)$, as shown in the graph in bottom part of Figure 2.1. (In the case of a binary variable $x \in \{0,1\}$, it is possible to represent this message by a log-likelihood ratio, which can be denoted by $\text{LLR}_{L\to R}(x)$.) It will be demonstrated in Section 2.2 that this extrinsic probability with respect to node L for the edge-variable x_0 is the correct quantity to use for the intrinsic probability for x_0 with respect to node R. Then the appropriate definition of the message vectors are

$$\begin{aligned} \overrightarrow{\mu}_{L\to R}(x_0) &= \overrightarrow{P}_L^{\text{ext}}(x_0) = \overrightarrow{P}_R^{\text{int}}(x_0) \\ \overrightarrow{\mu}_{R\to L}(x_0) &= \overrightarrow{P}_R^{\text{ext}}(x_0) = \overrightarrow{P}_L^{\text{int}}(x_0), \end{aligned}$$

so that the message vector $\overrightarrow{\mu}_{L\to R}(x_0)$ is used to carry the extrinsic probability output from node L to be used as the intrinsic probability in node R. Similarly, the message vector $\overrightarrow{\mu}_{R\to L}(x_0)$ passes the extrinsic probability from node R to be used as the intrinsic probability for node L.

By repeated computations at the nodes, and passing message vectors along the edges, this *message-passing algorithm* (MPA) yields probabilities that reflect the local constraints and the structure of the entire graph. This algorithm gives a way to solve difficult problems if they can be represented by a graph of relatively simple local constraints. Decoders based on this approach are also known as APP (*a posteriori* probability) decoders, since the final result is often a posterior probability. [Gal62]. (These decoders are also known in the literature as MAP (maximum *a posteriori*) decoders, which can be a misnomer since maximization does not always take place in the decoding process).

The predominant formulation of this algorithm in the literature has been the sum-product algorithm (SPA) for factor graphs (ref. Kschis-

chang *et al.* [KFL01] and Wiberg [Wib97]). Factor graphs are a generalization of Tanner graphs [Tan81], and consist of a bipartite graph between two types of nodes, function nodes and variable nodes. (Some of the variable nodes are known as state nodes). In comparison, normal graphs have only one type of node (similar to the function node), and variables are associated to the edges. It is shown in [For01] that with the proper transformations, a factor graph can be represented by a normal graph with essentially the same decoding complexity, so that the focus on normal graphs does not lessen the generality or efficiency of the algorithm.[5]

The sum-product algorithm is also known as the "two-way" algorithm (TWA) [For97]. The connection to belief propagation (BP) for Bayesian networks [Pea88] was found in McEliece *et al.* [MMC98]. Another formulation using the generalized distributive law (GDL) for functions is given by Aji and McEliece [AM00].

2. MESSAGE-PASSING ON GRAPHS

The basic principle of the *message-passing algorithm* is to break up a computationally difficult problem into a multitude of easier subproblems. It operates by iteratively passing probabilistic messages in a graph, and can be used to evaluate extrinsic and posterior probabilities based on intrinsic probabilities and the structure of the graph.

In this section, we consider message-passing on graphs by studying the situations of one node, two nodes and many nodes.[6] For a single node, the extrinsic probability for an edge-variable is based on the intrinsic probabilities on all the other edge-variables. For two nodes connected by an edge, it is shown that the extrinsic probability from one node serves as the intrinsic probability for the other. These results are then generalized for the case of many nodes, and the message-passing algorithm is derived.

2.1 ONE NODE

Consider a single node N, with $K+1$ edges, as shown in Figure 2.2. The edges are associated with the variables $x_0, x_1, ..., x_K$, which belong to the alphabets $A_0, A_1, ..., A_K$, respectively. Suppose that intrinsic probabilities with respect to node N are available for the variables $x_0, x_1, ..., x_K$. Recall that for each variable x_i, the intrinsic probability $P_N^{\text{int}}(x_i)$ denotes a probability vector of length $|A_i|$, with one entry $P_N^{\text{int}}(x_i = \xi_i)$ for each possible value $\xi_i \in A_i$. In addition, it is assumed that the variables are independent, so that the joint intrinsic probability $P_N^{\text{int}}\left(\{x_i = \xi_i\}_{i=0}^K\right)$ factors into the product of the individual intrinsic probabilities $\prod_{i=0}^K P_N^{\text{int}}(x_i = \xi_i)$.

[5] Normal graphs can be represented by factor graphs by inserting a variable node into every edge of the normal graph. On the other hand, to convert a factor graph into a normal graph, simply replace its variable nodes with an equality node, as defined in Section 3.1.

[6] Linguists report that there exist languages where the concept of numbers is limited to *one*, *two*, and *many*.

The posterior probability is a conditional probability $P(x_0 = \xi_0 \mid N)$ that can be written using Bayes' theorem (2.10) as

$$\overbrace{P(x_0 = \xi_0 \mid N)}^{\text{posterior}} = \frac{1}{P(N)} \overbrace{P(N \mid x_0 = \xi_0)}^{\text{prop. to extrinsic}} \overbrace{P(x_0 = \xi_0)}^{\text{intrinsic}}. \tag{2.15}$$

The intrinsic probability is

$$P_N^{\text{int}}(x_0 = \xi_0) = P(x_0 = \xi_0),$$

while the extrinsic probability is proportional to $P(N \mid x_0 = \xi_0)$, as in

$$P_N^{\text{ext}}(x_0 = \xi_0) = c'_{x_0} P(N \mid x_0 = \xi_0).$$

The posterior probability is given by

$$P_N^{\text{post}}(x_0 = \xi_0) = P(x_0 = \xi_0 \mid N).$$

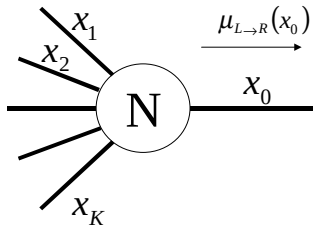

Figure 2.2. One node

2.1.1 EXTRINSIC PROBABILITY

Based on the intrinsic probabilities $P_N^{\text{int}}(x_0=\xi_0)$, $P_N^{\text{int}}(x_1=\xi_1)$, ..., $P_N^{\text{int}}(x_K=\xi_K)$, for all of the variables, we calculate the extrinsic probability for the variable x_0 with respect to the node N. In this case, the event N corresponds to the event that the values of the edge-variables $x_0, x_1, ..., x_K$ connected to the node N satisfy the local constraint, which is defined by a constraint set

$$S_N \subset A_0 \times A_1 \times \cdots \times A_K.$$

In other words, the event N is true exactly when the variables take on values $x_0=\xi_0$, $x_1=\xi_1$, ..., $x_K=\xi_K$, such that

$$(\xi_0, \xi_1, ..., \xi_K) \in S_N.$$

To evaluate the extrinsic probability, we evaluate the conditional probability $P(N \mid x_0=\xi_0)$. First, we note that the product of the alphabets $A_1 \times \cdots \times A_K$ gives a complete set of values for the variables $x_1, x_2, ..., x_K$, so that

$$\sum_{\substack{(\xi_1,...,\xi_K) \\ \in A_1 \times \cdots \times A_K}} P\left(\{x_i=\xi_i\}_{i=1}^K\right) = 1,$$

as in (2.1). Then it is possible to apply (2.2) to obtain the decomposition

$$P(N) = \sum_{\substack{(\xi_1,...,\xi_K) \\ \in A_1 \times \cdots \times A_K}} P\left(N, \{x_i=\xi_i\}_{i=1}^K\right).$$

The conditional probability $P(N \mid x_0=\xi_0)$ can thus be evaluated as

$$P(N \mid x_0=\xi_0) = \sum_{\substack{(\xi_1,...,\xi_K) \\ \in A_1 \times \cdots \times A_K}} P\left(N, \{x_i=\xi_i\}_{i=1}^K \mid x_0=\xi_0\right). \quad (2.16)$$

Applying the conditional probability rule (2.7) to each term in the summation gives

$$P\left(N, \{x_i=\xi_i\}_{i=1}^K \mid x_0=\xi_0\right) = P\left(N \mid \{x_i=\xi_i\}_{i=0}^K\right) \cdot P\left(\{x_i=\xi_i\}_{i=1}^K \mid x_0=\xi_0\right). \quad (2.17)$$

Then using the independence of the variables $\{x_i\}_{i=0}^K$, the conditioning on x_0 can be removed as in (2.5), and the joint probability can be factored as in (2.3), resulting in the expression

$$P\left(\{x_i=\xi_i\}_{i=1}^K \mid x_0=\xi_0\right) = \prod_{i=1}^K P(x_i=\xi_i) \quad (2.18)$$

for the second term on the right-hand side of (2.17).

On the other hand, for the first term on the right-hand side of (2.17), recall that the event N is true exactly when the value of the variables form a combination that belongs to the constraint set S_N. Then the conditional probability is given by

$$P\left(N \mid \{x_i=\xi_i\}_{i=0}^K\right) = \begin{cases} 1 & \text{if } (\xi_0, \xi_1, ..., \xi_K) \in S_N \\ 0 & \text{otherwise} \end{cases},$$

which can be expressed by an *indicator function*, denoted by square brackets.

$$P\left(N \mid \{x_i=\xi_i\}_{i=0}^K\right) = [(\xi_0, \xi_1, ..., \xi_K) \in S_N] \quad (2.19)$$

Then putting together (2.17), (2.19), and (2.18), the expression in (2.16) can be rewritten as

$$P(N \mid x_0=\xi_0) = \sum_{\substack{(\xi_1,...,\xi_K) \\ \in A_1 \times \cdots \times A_K}} [(\xi_0, \xi_1, ..., \xi_K) \in S_N] \prod_{i=1}^K P_N^{\text{int}}(x_i=\xi_i). \quad (2.20)$$

This use of the indicator function for the summation makes it clear that the constraint set S_N is used to select out certain combinations.

Finally, the indicator function can be incorporated in the range of the summation to end up with a compact expression for the conditional probability,

$$P(N \mid x_0=\xi_0) = \sum_{\substack{\xi_1,...,\xi_K \\ (\xi_0,\xi_1,...,\xi_K) \in S_N}} \prod_{i=1}^K P_N^{\text{int}}(x_i=\xi_i). \quad (2.21)$$

The extrinsic probability is then given by

$$\begin{aligned} P_N^{\text{ext}}(x_0=\xi_0) &= c'_{x_0} P(N \mid x_0=\xi_0) \\ &= c'_{x_0} \sum_{\substack{\xi_1,...,\xi_K \\ (\xi_0,\xi_1,...,\xi_K) \in S_N}} \prod_{i=1}^K P_N^{\text{int}}(x_i=\xi_i) \end{aligned} \quad (2.22)$$

where the normalization constant is

$$c'_{x_0} = \left(\sum_{\xi_0 \in A_0} P(N \mid x_0=\xi_0) \right)^{-1}.$$

This equation (2.22) is the key equation of the message-passing algorithm. With respect to a node, it gives the extrinsic probability for an

edge-variable in terms of the intrinsic probabilities for all the other edge-variables of the node. Some examples of nodes are the equality node and parity check node given in Sections 3.1 and 3.2.

2.1.2 POSTERIOR PROBABILITY

The posterior probability can be computed using (2.15), along with the expression for $P(N \mid x_0 = \xi_0)$ in (2.21):

$$\begin{aligned} P(x_0 = \xi_0 \mid N) &= \frac{1}{P(N)} P(N \mid x_0 = \xi_0)\, P_N^{\text{int}}(x_0 = \xi_0) \\ &= \frac{1}{P(N)} \sum_{\substack{\xi_1,\ldots,\xi_K \\ (\xi_0,\xi_1,\ldots,\xi_K)\in S_N}} \prod_{i=0}^{K} P_N^{\text{int}}(x_i = \xi_i) \end{aligned}$$

Introducing a normalization constant $\alpha_0 = (P(N))^{-1}$, the posterior probability can be rewritten as

$$P_N^{\text{post}}(x_0 = \xi_0) = \alpha_0 \sum_{\substack{\xi_1,\ldots,\xi_K \\ (\xi_0,\xi_1,\ldots,\xi_K)\in S_N}} \prod_{i=0}^{K} P_N^{\text{int}}(x_i = \xi_i)\,. \tag{2.23}$$

The probability $P(N)$ can be expanded as follows,

$$P(N) = \sum_{\xi_0 \in A} P(N \mid x_0 = \xi_0)\, P_N^{\text{int}}(x_0 = \xi_0)\,,$$

so that applying (2.21) defines the normalization constant,

$$\alpha_0 = (P(N))^{-1} = \left(\sum_{\substack{\xi_0,\xi_1,\ldots,\xi_K \\ (\xi_0,\xi_1,\ldots,\xi_K)\in S_N}} \prod_{i=0}^{K} P_N^{\text{int}}(x_i = \xi_i) \right)^{-1}.$$

This summation in (2.23) can also be understood as a *marginal* probability, where a function

$$g(\xi_0, \xi_1, \ldots, \xi_K) = [(\xi_0, \xi_1, \ldots, \xi_K) \in S_N] \prod_{i=0}^{K} P_N^{\text{int}}(x_i = \xi_i)$$

is summed over all variables except for one,

$$P_N^{\text{post}}(x_0 = \xi_0) = \alpha_0 \sum_{\xi_1,\ldots,\xi_K} g(\xi_0, \xi_1, \ldots, \xi_K)\,.$$

2.1.3 SIMPLIFIED NOTATION

It is convenient to introduce a *simplified notation* in which the variable x_i takes on a value that is left unstated in the equations but can be inferred from context. In particular, the expression $P(x_i)$ refers to the probability $P(x_i = \xi_i)$ that x_i equals some implicit value ξ_i.

Consider the following expression in simplified notation:

$$P(x_0) = \sum_{\substack{(x_0,x_1,x_2)\in S \\ \sim\{x_1\}}} P(x_0)\, P(x_1)\, P(x_2) \tag{2.24}$$

To interpret the simplified notation, first take any variable x_i and assign it to a value by replacing x_i with the expression $\{x_i = \xi_i\}$.

Next, the notation $\sim \{x_0\}$ is the *not-sum* notation (introduced in [KFL01]), which indicates that the summation is over all configurations of all variables, except for the variable(s) in the not-sum expression. In other words, in this case (2.24), the summation is over all combinations of $\{x_1 = \xi_1\}$ and $\{x_2 = \xi_2\}$ for $\xi_1 \in A_1$ and $\xi_2 \in A_2$, while x_0 is not included as a variable of summation. (Note that this notation is often redundant, since the not-summed variable can often be deduced by inspection, e.g., the variable has already been defined with a value outside the summation.)

Finally, the reference to the constraint set S can be changed to an indicator function. For example, the simplified expression (2.24) expands into the following expression,

$$P(x_0 = \xi_0) = \sum_{\substack{\xi_1\in A_1, \\ \xi_2\in A_2}} [(\xi_0, \xi_1, \xi_2) \in S]\, P(x_0 = \xi_0)\, P(x_1 = \xi_1)\, P(x_2 = \xi_2)\,, \tag{2.25}$$

where the summation is over the variables ξ_1 and ξ_2, with the restriction that the triple (ξ_0, ξ_1, ξ_2) belongs to the set S. This example illustrates how the simplified notation in (2.24) can be used to compactly represent the expression (2.25).

Applying this simplified notation to the expression for extrinsic probability for x_0 in (2.22) yields

$$P_N^{\text{ext}}(x_0) = \alpha'_{x_0} \sum_{\substack{(x_0,x_1,\ldots,x_K)\in S_N \\ \sim\{x_0\}}} \prod_{i=1}^{K} P_N^{\text{int}}(x_i)\,. \tag{2.26}$$

The posterior probability for x_0 in (2.23) can similarly be written in simplified notation:

$$P_N^{\text{post}}(x_0) = c_{x_0} \sum_{\substack{(x_0,x_1,\ldots,x_K)\in S_N \\ \sim\{x_0\}}} \prod_{i=0}^{K} P_N^{\text{int}}(x_i)\,. \tag{2.27}$$

It is understood that in both these expressions, (2.26) and (2.27), the probabilities are calculated for x_0 equal to some implied value ξ_0 (chosen from the alphabet A_0). In other words, the probability expression $P(x_0)$ defines a function in terms of the variable x_0, where it is possible to evaluate the function for any value ξ_0 of the variable x_0.

In addition, the summation is over all combinations of values for the variables $x_1 = \xi_1$, $x_2 = \xi_2$, up to $x_K = \xi_K$. In other words, except for variables that have already been assigned values (i.e., $x_0 = \xi_0$ in this case, defined on the left-hand side of the equation), the variables range in value over their respective alphabets,

$$\xi_1 \in A_1, \xi_2 \in A_2, \ldots, \xi_K \in A_K.$$

Finally, these values, along with $x_0 = \xi_0$, must satisfy the local constraint at N, so that the expression

$$(x_0, x_1, x_2, \ldots, x_K) \in S_N$$

in (2.26) and (2.27) really means that each combination of values must belong to the constraint set:

$$(\xi_0, \xi_1, \xi_2, \ldots, \xi_K) \in S_N$$

The expression for $P_N^{\text{ext}}(x_0)$ in (2.26) thus represents a function for calculating the extrinsic probability $P_N^{\text{ext}}(x_0 = \xi_0)$ for any value $\xi_0 \in A$.

These two expressions (2.26) and (2.27) generalize to any edge-variable of the node. For a node N with $K+1$ neighboring edge-variables $\{x_0, x_1, \ldots, x_K\}$, the extrinsic and posterior probabilities for x_i, based on the intrinsic probabilities for the edge-variables x_l for $0 \le l \le K$, where $l \ne i$, are as follows:

$$P_N^{\text{ext}}(x_i) = c'_{x_i} \sum_{\substack{(x_0,x_1,\ldots,x_K)\in S_N \\ \sim\{x_i\}}} \prod_{\substack{l=0 \\ l\ne i}}^{K} P_N^{\text{int}}(x_l) \tag{2.28}$$

$$P_N^{\text{post}}(x_i) = c_{x_i} \sum_{\substack{(x_0,x_1,\ldots,x_K)\in S_N \\ \sim\{x_i\}}} \prod_{l=0}^{K} P_N^{\text{int}}(x_l) \tag{2.29}$$

The constants c'_{x_i} and c_{x_i} are the appropriate normalization constants. In both cases, the variable x_i has a fixed value ξ_i defined on the left-hand side, and so the summation is over the other variables $\{x_l\}$ such that $l \ne i$. In addition, in (2.28), the intrinsic probability $P_N^{\text{int}}(x_i)$ for x_i is left out of the product.

2.2 TWO NODES

Suppose that there are two nodes L and R that are connected by an edge x_0. To these nodes are associated local constraints, described by sets S_L and S_R, respectively. In addition to the edge x_0, node L has K additional edges (labeled by $x_1^L, x_2^L, \ldots, x_K^L$) and node R has K' additional edges (labeled by $x_1^R, x_2^R, \ldots, x_{K'}^R$).

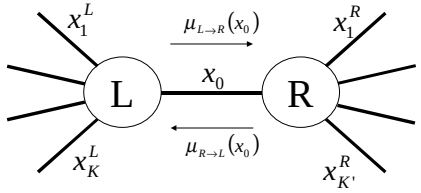

Figure 2.3. Two nodes

2.2.1 EXTRINSIC PROBABILITY WITH RESPECT TO THE GRAPH

Let the symbol $\mathcal{G}$ represent this graph, which includes two nodes and $K + K'$ external edges and 1 internal edge, as shown in Figure 2.3. It is assumed that there are intrinsic probabilities for the external edge-variables $(x_1^L, x_2^L, \ldots, x_K^L, x_1^R, x_2^R, \ldots, x_{K'}^R)$. The intrinsic probabilities with respect to the graph $\mathcal{G}$ take the form,

$$P_{\mathcal{G}}^{\text{int}}(x_i^L) = P(x_i^L) \text{ for } i = 1, 2, \ldots, K$$
$$P_{\mathcal{G}}^{\text{int}}(x_i^R) = P(x_i^R) \text{ for } i = 1, 2, \ldots, K'$$
$$P_{\mathcal{G}}^{\text{int}}(x_0) = \frac{1}{|A_0|}$$

where the intrinsic probability w.r.t. $\mathcal{G}$ for the internal edge x_0 is set to a uniform probability, e.g., $P_{\mathcal{G}}^{\text{int}}(x_0 = \xi_0) = \frac{1}{|A_0|}$ for all $\xi_0 \in A_0$.

The constraint set $S_{\mathcal{G}}$ for the graph consists of all configurations of the $K + K' + 1$ variables

$$(x_0, x_1^L, x_2^L, ..., x_K^L, x_1^R, x_2^R, ..., x_{K'}^R)$$

such that the appropriate subset of these values satisfy the constraint sets for both nodes L and R. Then membership in the constraint set $S_{\mathcal{G}}$ can be expressed by an indicator function, which factors into the product of indicator functions of the two nodes.

$$\begin{aligned}\left[(x_0, x_1^L, x_2^L, ..., x_K^L, x_1^R, x_2^R, ..., x_{K'}^R) \in S_{\mathcal{G}}\right] = \left[(x_0, x_1^L, x_2^L, ..., x_K^L) \in S_L\right] \\ \cdot \left[(x_0, x_1^R, x_2^R, ..., x_{K'}^R) \in S_R\right]\end{aligned} \tag{2.30}$$

Then with the definition of the intrinsic probabilities and a constraint set $S_{\mathcal{G}}$, it is possible to find the extrinsic probabilities, as in (2.28). Suppose that we rename all of the variables by

$$(x_0, x_1, x_2, ..., x_{K+K'}) = (x_0, x_1^L, x_2^L, ..., x_K^L, x_1^R, x_2^R, ..., x_{K'}^R)\,.$$

Then as in (2.28), the extrinsic probability for any edge-variable can be found from the intrinsic probabilities for all the other edge-variables.

$$P_{\mathcal{G}}^{\text{ext}}(x_i) = c'_{x_i} \sum_{\substack{(x_0, x_1, ..., x_{K+K'}) \in S_{\mathcal{G}} \\ \sim \{x_i\}}} \prod_{\substack{l=0 \\ l \neq i}}^{K+K'} P_{\mathcal{G}}^{\text{int}}(x_l) \tag{2.31}$$

This expression can be complicated since the number of edge-variables that vary in the summation is $K + K'$. It turns out, as we will see, that it is possible to use the constraint sets S_L and S_R to compute the extrinsic probabilities for each node separately, in such a way that the final results are the same as (2.31).

2.2.2 MARKOV PROPERTY

We first verify that the events L, $\{x_0 = \xi_0\}$ and R, in that order, form a Markov chain. In other words, we will show that

$$P(L, R \mid x_0) = P(L \mid x_0)\, P(R \mid x_0)\,. \tag{2.32}$$

The proof follows from expanding in terms of the other variables. For notational convenience, let $z^L = \{x_i^L\}_{i=1}^K$ and $z^R = \{x_i^R\}_{i=1}^{K'}$. Then

$$\begin{aligned}P(L, R \mid x_0) &= \sum_{z^L, z^R} P\left(L, R, z^L, z^R \mid x_0\right) \\ &= \sum_{z^L, z^R} P\left(L, R \mid x_0, z^L, z^R\right) P\left(z^L, z^R \mid x_0\right). \end{aligned} \tag{2.33}$$

The first term in the right-hand side of (2.33) is given by an indicator function that indicates whether the configuration belongs to the constraint set $S_{\mathcal{G}}$, which can be factored using (2.30):

$$\begin{aligned}P\left(L, R \mid x_0, z^L, z^R\right) &= \left[(x_0, z^L, z^R) \in S_{\mathcal{G}}\right] \\ &= \left[(x_0, z^L) \in S_L\right] \left[(x_0, z^R) \in S_R\right] \\ &= P\left(L \mid x_0, z^L\right) P\left(R \mid x_0, z^R\right)\end{aligned} \tag{2.34}$$

Meanwhile, the second term in the right-hand side of (2.33) reduces to

$$P\left(z^L, z^R \mid x_0\right) = P\left(z^L\right) P\left(z^R\right) \tag{2.35}$$

since the variables are independent. Putting (2.34) and (2.35) together in (2.33) gives

$$P(L, R \mid x_0) = \sum_{z^L, z^R} P\left(L \mid x_0, z^L\right) P\left(z^L\right) P\left(R \mid x_0, z^R\right) P\left(z^R\right),$$

so that by an application of the Cartesian-product distributive law (which is generalized in [AM00]), we have

$$\begin{aligned}P(L, R \mid x_0) &= \left(\sum_{z^L} P\left(L \mid x_0, z^L\right) P\left(z^L\right)\right) \left(\sum_{z^R} P\left(R \mid x_0, z^R\right) P\left(z^R\right)\right) \\ &= P(L \mid x_0)\, P(R \mid x_0),\end{aligned}$$

which was to be demonstrated.

2.2.3 COMPUTING THE EXTRINSIC PROBABILITY FOR AN EXTERNAL EDGE

Let us consider the extrinsic probability for x_1^R w.r.t. $\mathcal{G}$, which is given by

$$P_{\mathcal{G}}^{\text{ext}}\left(x_1^R\right) = c'_{x_1^R} P\left(L, R \mid x_1^R\right)$$

for some normalization constant $c'_{x_1^R}$. For notational simplicity, let

$$\left\{x_i^R\right\}_{i=2}^{K'} = \{x_2^R, x_3^R, ..., x_{K'}^R\}$$

represent the remaining edge-variables of R besides x_0 and x_1^R. Then the conditional probability can be expanded as

$$\begin{aligned} P\left(L, R \mid x_1^R\right) &= \sum_{x_0} P\left(L, R, x_0, y^R \mid x_1^R\right) \\ &= \sum_{\substack{x_0, \\ \{x_i^R\}_{i=1}^{K'}}} P\left(R \mid L, x_0, \{x_i^R\}_{i=1}^{K'}\right) P\left(L, x_0, \{x_i^R\}_{i=2}^{K'} \mid x_1^R\right). \end{aligned} \tag{2.36}$$

Due to the Markovity of L, x_0, and R, the dependence on L can be dropped from the conditional probability in (2.36) since the expression is already conditioned on x_0.

$$P\left(R \mid L, x_0, \{x_i^R\}_{i=1}^{K'}\right) = P\left(R \mid x_0, x_1^R, x_2^R, ..., x_{K'}^R\right) \tag{2.37}$$

In addition, for the second part of (2.36), some of the dependencies can be removed due to the Markov property and the independence of the variables.

$$\begin{aligned} P\left(L, x_0, \{x_i^R\}_{i=2}^{K'} \mid x_1^R\right) &= P\left(L \mid x_0, \{x_i^R\}_{i=1}^{K'}\right) P\left(x_0, \{x_i^R\}_{i=2}^{K'} \mid x_1^R\right) \\ &= P\left(L \mid x_0\right) P_{\mathcal{G}}^{\text{int}}\left(x_0\right) \prod_{i=2}^{K'} P_{\mathcal{G}}^{\text{int}}\left(x_i^R\right) \end{aligned} \tag{2.38}$$

Since $P(x_0)$ is the intrinsic probability w.r.t. the graph $\mathcal{G}$, recall that it was set to a uniform distribution $P_{\mathcal{G}}^{\text{int}}(x_0 = \xi_0) = \frac{1}{|A_0|}$, which can then be absorbed into the normalization constant.

Putting (2.37) and (2.38) into (2.36) gives

$$\begin{aligned} P_{\mathcal{G}}^{\text{ext}}\left(x_1^R\right) &= c'_{x_1^R} P\left(L, R \mid x_1^R\right) \\ &= c''_{x_1^R} \sum_{x_0, x_2^R, ..., x_{K'}^R} P\left(R \mid x_0, x_1^R, ..., x_{K'}^R\right) P\left(L \mid x_0\right) \prod_{i=2}^{K'} P_{\mathcal{G}}^{\text{int}}\left(x_i^R\right) \\ &= c''_{x_1^R} \sum_{\substack{\left(x_0, x_1^R, x_2^R, ..., x_{K'}^R\right) \in S_R \\ \sim \{x_1^R\}}} P\left(L \mid x_0\right) \prod_{i=2}^{K'} P_{\mathcal{G}}^{\text{int}}\left(x_i^R\right), \end{aligned} \tag{2.39}$$

where $c''_{x_1^R}$ is the appropriate normalization constant. In comparison, the analogous expression for extrinsic probability w.r.t. the node R is given

by

$$P_R^{\text{ext}}\left(x_1^R\right) = c'_{x_1^R} \sum_{\substack{\left(x_0, x_1^R, x_2^R, ..., x_{K'}^R\right) \in S_R \\ \sim \{x_1^R\}}} P_R^{\text{int}}\left(x_0\right) \prod_{i=2}^{K'} P_R^{\text{int}}\left(x_i^R\right), \tag{2.40}$$

which is a direct application of (2.28).

Comparing these two equations shows that using the node computation (2.40) with an input of $P_R^{\text{int}}(x_0)$ set to $P(L \mid x_0)$ gives an identical calculation to the extrinsic probability of the two-node graph in (2.39). Hence, this suggests the following set of assignments,

$$\begin{aligned} P_R^{\text{int}}\left(x_i^R\right) &\leftarrow P_{\mathcal{G}}^{\text{int}}\left(x_i^R\right) \\ P_L^{\text{int}}\left(x_i^L\right) &\leftarrow P_{\mathcal{G}}^{\text{int}}\left(x_i^L\right) \\ P_R^{\text{int}}\left(x_0\right) &\leftarrow P_L^{\text{ext}}\left(x_0\right) \\ P_L^{\text{int}}\left(x_0\right) &\leftarrow P_R^{\text{ext}}\left(x_0\right) \end{aligned} \tag{2.41}$$

where the extrinsic probability $P_L^{\text{ext}}(x_0)$ is simply the normalized version of $P(L \mid x_0)$.

In other words, we start with intrinsic probabilities w.r.t. $\mathcal{G}$, and then assign them to the nodes, so that $P_L^{\text{int}}\left(x_i^L\right) = P_{\mathcal{G}}^{\text{int}}\left(x_i^L\right)$. With these intrinsic probabilities w.r.t. L, the extrinsic probability for x_0 w.r.t. L is given by

$$P_L^{\text{ext}}\left(x_0\right) = c'_{x_0} \sum_{\substack{\left(x_0, x_1^L, ..., x_K^L\right) \in S_L \\ \sim \{x_0\}}} \prod_{i=1}^{K} P_L^{\text{int}}\left(x_i^L\right),$$

as in (2.26). Then if this extrinsic probability $P_L^{\text{ext}}(x_0)$ is used as the intrinsic probability $P_R^{\text{int}}(x_0)$ in (2.40), the resulting extrinsic probability w.r.t. R turns out to also be the extrinsic probability w.r.t. the graph $\mathcal{G}$, according to (2.39), i.e., $P_{\mathcal{G}}^{\text{ext}}\left(x_1^R\right) = P_R^{\text{ext}}\left(x_1^R\right)$.

2.2.4 EXTRINSIC FROM ONE NODE IS INTRINSIC FOR THE OTHER

The choice of the edge-variable x_1^R in (2.39) and (2.40) was arbitrary. Starting with the assignments in (2.41), and applying the update rule (2.28) to each edge-variable connected to nodes L and R, the computations at the two nodes yield extrinsic probabilties that are also the

extrinsic probabilities w.r.t. the graph $\mathcal{G}$:

$$P_{\mathcal{G}}^{\text{int}}\left(x_i^L\right) \leftarrow P_L^{\text{ext}}\left(x_i^L\right) \text{ for } i = 1, 2, ..., K$$
$$P_{\mathcal{G}}^{\text{int}}\left(x_i^R\right) \leftarrow P_R^{\text{ext}}\left(x_i^R\right) \text{ for } i' = 1, 2, ..., K'$$

Then it is not necessary to use (2.31) to evaluate the extrinsic probability, but instead computations can be made at both nodes. For the case of binary variables, this entails two summations of up to 2^K and $2^{K'}$ terms, for nodes L and R, respectively, rather than one summation over $2^{K+K'}$ terms when considering the entire graph. Breaking up the computation in this way can thus reduce the complexity.

Given the assignments in (2.41), we can define messages $\mu_{L\to R}(x_0)$ and $\mu_{R\to L}(x_0)$ between the nodes, as shown in Figure 2.3. The notation $\mu_{L\to R}(x_0)$ indicates that it is a message from node L to node R along the edge associated with the variable x_0. (In analogy to the probability $P(x_0)$, the notation $\mu_{L\to R}(x_0)$ represents a value $\mu_{L\to R}(x_0 = \xi_0)$ for some implicit value ξ_0, using the simplified notation in Section 2.1.3.) The messages carry the extrinsic probability from one node to be used as the intrinsic probability at the other node.

$$\mu_{L\to R}(x_0) = P_L^{\text{ext}}(x_0) = P_R^{\text{int}}(x_0)$$
$$\mu_{R\to L}(x_0) = P_R^{\text{ext}}(x_0) = P_L^{\text{int}}(x_0)$$

In short, *the extrinsic probability for one node gives the intrinsic probability for the other node*, and the messages $\mu_{L\to R}(x_0)$ and $\mu_{R\to L}(x_0)$ serve to pass these probabilities between the two nodes. The basic idea is to use the computation based on the local constraints at the nodes as subroutines in the main program for finding the extrinsic probabilities w.r.t. $\mathcal{G}$. The relationship of the various intrinsic and extrinsic probabilities is also shown in Figure 2.4.

Message-passing for two nodes

- Step 0. **Initialize**

  Input: $P_{\mathcal{G}}^{\text{int}}\left(x_i^L\right)$, for $i = 1, 2, ..., K$, and $P_{\mathcal{G}}^{\text{int}}\left(x_{i'}^R\right)$, for $i' = 1, 2, ..., K'$
  Internal edge-variable set to uniform: $P_{\mathcal{G}}^{\text{int}}(x_0 = \xi_0) = \frac{1}{|A_0|}$

- Step 1a. **Compute message from L to R**

  Input: $P_L^{\text{int}}\left(x_i^L\right) \leftarrow P_{\mathcal{G}}^{\text{int}}\left(x_i^L\right)$, for $i = 1, 2, ..., K$

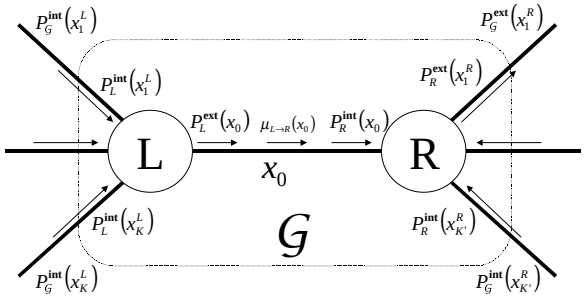

Figure 2.4. Message-passing for two nodes

Computation:

$$P_L^{\text{ext}}(x_0) = c'_{x_0} \sum_{\substack{(x_0, x_1^L, ..., x_K^L) \in S_L \\ \sim\{x_0\}}} \prod_{i=1}^{K} P_L^{\text{int}}\left(x_i^L\right)$$

Output: $\mu_{L\to R}(x_0) \leftarrow P_L^{\text{ext}}(x_0)$

- Step 1b. **Compute message from R to L**

  Input: $P_R^{\text{int}}\left(x_{i'}^R\right) \leftarrow P_{\mathcal{G}}^{\text{int}}\left(x_{i'}^R\right)$, for $i' = 1, 2, ..., K'$
  Computation:

$$P_R^{\text{ext}}(x_0) = c'_{x_0} \sum_{\substack{(x_0, x_1^R, ..., x_{K'}^R) \in S_R \\ \sim\{x_0\}}} \prod_{i=1}^{K'} P_R^{\text{int}}\left(x_{i'}^R\right)$$

  Output $\mu_{R\to L}(x_0) \leftarrow P_R^{\text{ext}}(x_0)$

- Step 2a. **Compute extrinsic output for L**

  Input: $P_L^{\text{int}}(x_0) \leftarrow \mu_{R\to L}(x_0)$ and $P_L^{\text{int}}\left(x_i^L\right) \leftarrow P_{\mathcal{G}}^{\text{int}}\left(x_i^L\right)$

Computation: for $i = 1, 2, ..., K$,

$$P_L^{\text{ext}}\left(x_i^L\right) = c'_{x_i^L} \sum_{\substack{\left(x_0, x_1^L, ..., x_K^L\right) \in S_L \\ \sim\left\{x_i^L\right\}}} P_L^{\text{int}}\left(x_0\right) \prod_{\substack{l=1 \\ l \neq i}}^{K} P_L^{\text{int}}\left(x_l^L\right)$$

Output: $P_{\mathcal{G}}^{\text{ext}}\left(x_i^L\right) \leftarrow P_L^{\text{ext}}\left(x_i^L\right)$ for $i = 1, 2, ..., K$

- Step 2b. **Compute extrinsic output for** R

Input: $P_R^{\text{int}}\left(x_0\right) \leftarrow \mu_{L \to R}\left(x_0\right)$, and $P_R^{\text{int}}\left(x_{i'}^R\right) \leftarrow P_{\mathcal{G}}^{\text{int}}\left(x_{i'}^R\right)$
Computation: for $i' = 1, 2, ..., K'$,

$$P_R^{\text{ext}}\left(x_{i'}^R\right) = c'_{x_{i'}^R} \sum_{\substack{\left(x_0, x_1^R, ..., x_{K'}^R\right) \in S_R \\ \sim\left\{x_{i'}^R\right\}}} P_R^{\text{int}}\left(x_0\right) \prod_{\substack{l=1 \\ l \neq i'}}^{K'} P_R^{\text{int}}\left(x_l^R\right)$$

Output: $P_{\mathcal{G}}^{\text{ext}}\left(x_{i'}^R\right) \leftarrow P_R^{\text{ext}}\left(x_{i'}^R\right)$ for $i' = 1, 2, ..., K'$

It should be mentioned that another consequence of the assignments in (2.41) is that the posterior probability expressions for x_0 become consistent. In other words, the posterior probability w.r.t. the graph $\mathcal{G}$ is equal to the posterior probability w.r.t. the node L and w.r.t. the node R: Using Bayes' theorem and (2.32), we have

$$\begin{aligned} P_{\mathcal{G}}^{\text{post}}\left(x_0\right) &= P\left(x_0 \mid L, R\right) \\ &= \frac{P\left(x_0\right)}{P\left(L, R\right)} P\left(L, R \mid x_0\right) \\ &= \frac{P\left(x_0\right)}{P\left(L, R\right)} P\left(L \mid x_0\right) P\left(R \mid x_0\right). \end{aligned}$$

Since $P\left(x_0\right) = P_{\mathcal{G}}^{\text{int}}\left(x_0\right) = \frac{1}{|A_0|}$, and $P\left(L \mid x_0\right)$ and $P\left(R \mid x_0\right)$ are proportional to their respective extrinsic probabilities, we have

$$P_{\mathcal{G}}^{\text{post}}\left(x_0\right) = c_{x_0} P_L^{\text{ext}}\left(x_0\right) P_R^{\text{ext}}\left(x_0\right). \tag{2.42}$$

On the other hand, from the basic relationship of intrinsic, extrinsic and posterior probabilities in (2.12), we have

$$P_L^{\text{post}}\left(x_0\right) = c_{x_0} P_L^{\text{ext}}\left(x_0\right) P_L^{\text{int}}\left(x_0\right) \tag{2.43}$$

$$P_R^{\text{post}}\left(x_0\right) = c_{x_0} P_R^{\text{int}}\left(x_0\right) P_R^{\text{ext}}\left(x_0\right). \tag{2.44}$$

With the assignment of $P_L^{\text{int}}\left(x_0\right) = P_R^{\text{ext}}\left(x_0\right)$ and $P_R^{\text{int}}\left(x_0\right) = P_L^{\text{ext}}\left(x_0\right)$ in (2.41), these three expressions (2.42), (2.43) and (2.44) for the posterior probability become identical. This observation lends additional credence to the choice of the assignments in (2.41).

2.3 MANY NODES

This example of two nodes connected by a single edge can be generalized to a graph of many nodes, known as a module.

2.3.1 ONE MODULE

A *module* consists of an arbitrary normal graph $\mathcal{G}$, as defined in Section 1.2. The set of nodes $N_{\mathcal{G}}$ is composed of nodes N_j for $j = 1, ..., |N_{\mathcal{G}}|$, while the set of edges $E_{\mathcal{G}}$ have associated variables x_i for $i = 1, ..., |E_{\mathcal{G}}|$, where the variable x_i belongs to some alphabet A_i. An example with $|N_{\mathcal{G}}| = 8$ and $|E_{\mathcal{G}}| = 11$ is shown in Figure 2.5. Each node N_j is connected to some subset of the edges, and edges can be connected to one or two (distinct) nodes. An edge connected to two nodes is an *internal edge*, while an edge connected to only one node is an *external edge* of the module. The external edges of the module are used for passing intrinsic probabilities $P_{\mathcal{G}}^{\text{int}}\left(x_i\right)$ into the module. On the other hand, the intrinsic probabilities w.r.t. $\mathcal{G}$ for the internal edges are set by default to uniform distributions $P_{\mathcal{G}}^{\text{int}}\left(x_{i'} = a\right) = \frac{1}{|A_{i'}|}$ for all $a \in A_{i'}$. In a sense, the module can be view as a big node whose edges are given by the external edges of the module.

Recall that each node has a local constraint, which selects a subset of the possible configurations of the neighboring edge-variables. As for the constraint set for the module, a configuration of values for edge-variables in $E_{\mathcal{G}}$ is considered valid for the entire graph $\mathcal{G}$ if the configuration also satisfies all of the local constraints at the nodes in $N_{\mathcal{G}}$. Let E_{N_j} represent the set of neighboring edges for the node N_j in the graph. Then the constraint set S_{N_j} defines a subset of valid configurations out of the direct product of the alphabets A_i for $x_i \in E_{N_j}$. In other words, the set $S_{\mathcal{G}}$ is composed of configurations

$$\left\{\xi_1, ..., \xi_{|E_{\mathcal{G}}|}\right\} \in A_1 \times \cdots \times A_{|E_{\mathcal{G}}|}$$

such that for each nodes N_j (for $j = 1, 2, ..., |N_{\mathcal{G}}|$), the subset of the configuration corresponding to E_{N_j} satisfies the constraint set S_{N_j}. Hence, the local constraints of the nodes combine to define an overall constraint on the edge-variables.

Using this constraint set $S_{\mathcal{G}}$, it becomes possible to calculate the extrinsic (and posterior) probabilities for all the external edge-variables

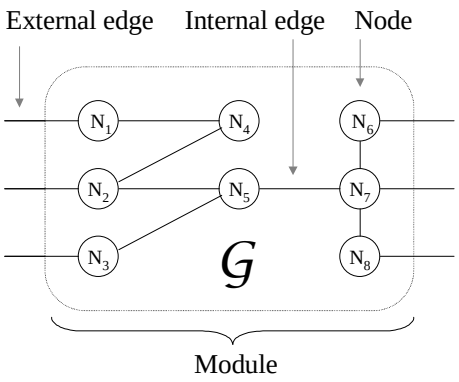

Figure 2.5. An example of a module

based on the intrinsic probabilities. With this definition of the constraint set $S_{\mathcal{G}}$, we can compute the extrinsic probability as

$$P_{\mathcal{G}}^{\text{ext}}(x_i) = c'_{x_i} \sum_{\substack{(x_1,\ldots,x_{|E_{\mathcal{G}}|}) \in S_{\mathcal{G}} \\ \sim\{x_i\}}} \prod_{l=1, l\neq i}^{|E_{\mathcal{G}}|} P_{\mathcal{G}}^{\text{int}}(x_l)\,, \tag{2.45}$$

where c'_{x_i} is the appropriate constant. (For the posterior probability, we need to multiply by $P_{\mathcal{G}}^{\text{int}}(x_i)$ and normalize again.) In general, when there are many nodes and edges in the graph $\mathcal{G}$, this expression can be quite complicated to evaluate.

2.3.2 TWO MODULES

In the case that the graph $\mathcal{G}$ can be divided into two subgraphs sharing a single common edge x_0, it is possible to divide the problem in (2.45) into two subproblems, and pass messages between them. Label the two subgraphs by $\mathcal{L}$ and $\mathcal{R}$, for left and right, respectively, as shown in Figure 2.6. The set of nodes $N_{\mathcal{G}}$ is the disjoint union of the sets $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$. Meanwhile, the set of edges $E_{\mathcal{G}}$ is the union of $E_{\mathcal{L}}$ and $E_{\mathcal{R}}$, where the

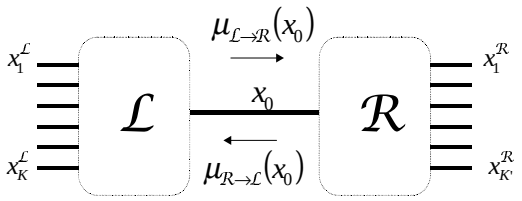

Figure 2.6. Two modules

intersection consists of a single edge x_0.

$$\begin{aligned} N_{\mathcal{L}} \cup N_{\mathcal{R}} &= N_{\mathcal{G}} \\ N_{\mathcal{L}} \cap N_{\mathcal{R}} &= \emptyset \\ E_{\mathcal{L}} \cup E_{\mathcal{R}} &= E_{\mathcal{G}} \\ E_{\mathcal{L}} \cap E_{\mathcal{R}} &= \{x_0\} \end{aligned}$$

It is convenient to rename the edge-variables in $E_{\mathcal{G}}$ based on the subgraph that they belong to. In addition to the common edge x_0, let the module $\mathcal{L}$ contain the edge-variables $x_1^{\mathcal{L}},\ldots,x_K^{\mathcal{L}}$, and the module $\mathcal{R}$ contain the edge-variables $x_1^{\mathcal{R}},\ldots,x_{K'}^{\mathcal{R}}$, so that $|E_{\mathcal{L}}| = K+1$ and $|E_{\mathcal{R}}| = K'+1$. Note that if we cut the graph at the edge x_0, then the graph becomes decomposed into two modules, and the internal edge x_0 becomes two external edges (also known as half-edges). In Figure 2.7, the module $\mathcal{G}$ from Figure 2.5 is divided into two modules $\mathcal{L}$ and $\mathcal{R}$, with $|N_{\mathcal{L}}| = 5$, $|E_{\mathcal{L}}| = 8$, $|N_{\mathcal{R}}| = 3$, and $|E_{\mathcal{R}}| = 4$.

With these definitions of the modules, we can derive similar expressions as (2.39) and (2.40) if we replace L and R by their calligraphic equivalents $\mathcal{L}$ and $\mathcal{R}$. From this we conclude that as in the two-node situation, setting $P_{\mathcal{R}}^{\text{int}}(x_0)$ to $P(\mathcal{L} \mid x_0)$ is the appropriate choice to make the extrinsic probabilities w.r.t. $\mathcal{R}$ equal to the extrinsic probabilities w.r.t. the whole graph $\mathcal{G}$. Hence, this suggests the following set of assignments,

$$\begin{aligned} P_{\mathcal{R}}^{\text{int}}(x_i^{\mathcal{R}}) &\leftarrow P_{\mathcal{G}}^{\text{int}}(x_i^{\mathcal{R}}) \\ P_{\mathcal{L}}^{\text{int}}(x_i^{\mathcal{L}}) &\leftarrow P_{\mathcal{G}}^{\text{int}}(x_i^{\mathcal{L}}) \\ P_{\mathcal{R}}^{\text{int}}(x_0) &\leftarrow P_{\mathcal{L}}^{\text{ext}}(x_0) \\ P_{\mathcal{L}}^{\text{int}}(x_0) &\leftarrow P_{\mathcal{R}}^{\text{ext}}(x_0) \end{aligned} \tag{2.46}$$

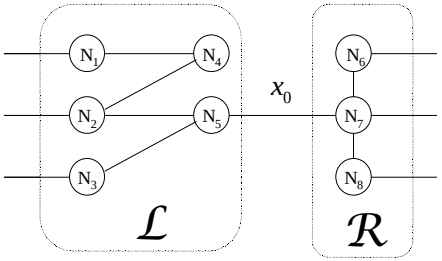

Figure 2.7. Two modules sharing an edge

where the extrinsic probability $P_{\mathcal{L}}^{\text{ext}}(x_0)$ is simply the normalized version of $P(\mathcal{L} \mid x_0)$.

In other words, to leverage the computations based on module $\mathcal{L}$ and module $\mathcal{R}$ for the combined graph $\mathcal{G}$, we should pass the following messages between the two modules, in exact analogy to the case of two nodes:

$$\mu_{\mathcal{L}\to\mathcal{R}}(x_0) = P_{\mathcal{L}}^{\text{ext}}(x_0) = P_{\mathcal{R}}^{\text{int}}(x_0)$$
$$\mu_{\mathcal{R}\to\mathcal{L}}(x_0) = P_{\mathcal{R}}^{\text{ext}}(x_0) = P_{\mathcal{L}}^{\text{int}}(x_0).$$

The extrinsic probability for x_0 from module $\mathcal{L}$ gives the prior probability for module $\mathcal{R}$, and the extrinsic probability for module $\mathcal{R}$ gives the intrinsic probability for module $\mathcal{L}$. In short, for a single edge connecting two modules, *the extrinsic probability from one module gives the intrinsic probability for the other module.*

Generally speaking, any message along an edge is extrinsic information from the node (or module) whence it came, and intrinsic information for the node (or module) to which it goes. By performing computations separately for the modules $\mathcal{L}$ and $\mathcal{R}$, and then passing messages $\mu_{\mathcal{L}\to\mathcal{R}}(x_0)$ and $\mu_{\mathcal{R}\to\mathcal{L}}(x_0)$, it is possible to solve the larger problem of computing the overall extrinsic probabilities for the combined graph $\mathcal{G}$. The relation of the various messages are shown in Figure 2.8, (which is the same as Figure 2.4, except that the node has been replaced by a module). The procedure for two modules is exactly an replica of the

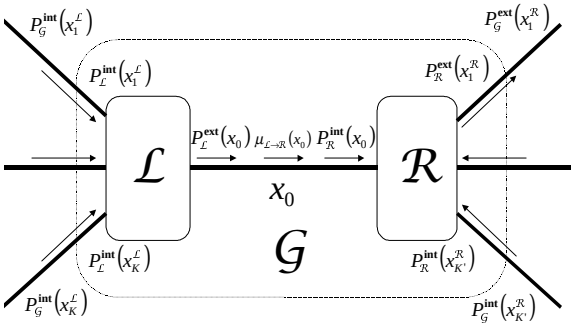

Figure 2.8. Message-passing between two modules

procedure for two nodes, so that the "program" in Section 2.2.4 for two nodes L and R can be followed exactly for the two modules $\mathcal{L}$ and $\mathcal{R}$.

2.4 MESSAGE-PASSING ALGORITHM

A *cycle* is a walk along the edges of a graph that begins and ends at the same node, without traversing any edge more than once. A graph without cycles is a *tree* (or is composed of multiple trees). We first consider the message-passing algorithm for graphs without cycles, and then talk about its application to graphs with cycles.

2.4.1 GRAPHS WITHOUT CYCLES

For a cycle-free graph, a cut at any edge divides the graph into two disconnected subgraphs. From this observation, we can use the results in the previous sections to derive the message-passing algorithm. Starting with a graph $\mathcal{G}$, we pick an arbitrary internal edge x_i, which divides the graph into two modules $\mathcal{L}$ and $\mathcal{R}$. Then the extrinsic probabilities can be calculated for the graph $\mathcal{G}$ by computing them for modules $\mathcal{L}$ and $\mathcal{R}$ and passing the appropriate messages along the edge x_i:

$$\mu_{\mathcal{L}\to\mathcal{R}}(x_i) = P_{\mathcal{L}}^{\text{ext}}(x_i) = P_{\mathcal{R}}^{\text{int}}(x_i)$$
$$\mu_{\mathcal{R}\to\mathcal{L}}(x_i) = P_{\mathcal{R}}^{\text{ext}}(x_i) = P_{\mathcal{L}}^{\text{int}}(x_i).$$

Proceeding recursively, we can then take each of these modules in turn (e.g., $\mathcal{L}$), and choose an internal edge (e.g., $x_j^{\mathcal{L}}$) which splits that module (i.e., $\mathcal{L}$) into two smaller modules (e.g., labeled $\mathcal{LL}$ and $\mathcal{LR}$). By passing the appropriate messages in both directions along that internal edge (i.e., $x_j^{\mathcal{L}}$), the extrinsic probabilities for the module (i.e., $\mathcal{L}$) can be found by computing the extrinsic probabilities for the two smaller modules (i.e., $\mathcal{LL}$ and $\mathcal{LR}$). Taking this recursive principle to its logical conclusion gives the result that the exact extrinsic probabilities for the graph $\mathcal{G}$ can be obtained by performing computations for each individual node and then passing messages between the nodes. The proof of exactness for cycle-free graphs is also found in [Wib97][KFL01][For01].

To recap this procedure, the input to the graph is a collection of intrinsic probabilities for the external edges-variables $P_{\mathcal{G}}^{\mathbf{int}}(x_i)$. Meanwhile, internal edge-variables start off by default with uniform intrinsic probabilities $P_{\mathcal{G}}^{\mathbf{int}}(x_{i'}) = \frac{1}{|A_{i'}|}$. (Note that when the graph contains cycles, then it is also necessary to initialize messages in both directions along all internal edges, as discussed in Section 2.4.2.) For each node N_j, the extrinsic probabilities with respect to that node can be found from the intrinsic probabilities with respect to that node using equation (2.28), which is duplicated below in (2.47). This expression is also known as the *sum-product update rule*, since this computation consists of the sum of products. As a result, the message-passing algorithm is also known as the *sum-product algorithm*.

The sum-product update rule is repeatly applied for the nodes, and messages are passed along the edges. The order in which the messages are passed is referred to as the *message-passing schedule*. The *stopping criterion* is the test that determines whether to stop passing messages and declare that the decoding is complete. The output from this algorithm can then be read as the extrinsic probabilities for the external edges. (For some applications, the final desired answer may be the posterior probability, which can be easily calculated from putting together the intrinsic and extrinsic probabilities for the external edge-variables using (2.12).)

The *message-passing algorithm* can be summarized as follows:

Message-passing algorithm

- Step 0. **Initialize.** External edge-variables start off with input messages corresponding to the intrinsic probability w.r.t. the graph. (In graphs with cycles, the internal edge-variables also start off with default messages in both directions.)

- Step 1. **Sum-product update rule.** For a node N that is connected to $K+1$ nodes, $X_0, X_1, ..., X_K$ (via the edge-variables $x_0, x_1, ..., x_K$, respectively), the output message $\mu_{N \to X_i}(x_i)$ for one edge is found from the input messages $\mu_{X_l \to N}(x_l)$ along the other K edges ($l = 0, 1, 2, ..., K$ and $l \neq i$).

$$\mu_{N \to X_i}(x_i) = c_{x_i} \sum_{\substack{(x_0, x_1, ..., x_K) \in S_N \\ \sim \{x_i\}}} \prod_{\substack{l=0 \\ l \neq i}}^{K} \mu_{X_l \to N}(x_l) \tag{2.47}$$

  The update formula applies for all $i = 0, 1, ..., K$.

- Step 2. **Message-passing.** For each edge, the output message from one node is passed along the edge to become the input message for the other node. Repeat Step 1 for all nodes according to the *message-passing schedule* until the *stopping criterion* has been reached.

More generally, the term "message-passing algorithm" can refer to other distributed computations on graphs involving the passing of messages, but we use it to refer specifically to the case where the computations are based on the sum-product update rule (2.47). Some other examples are soldier-counting as in Chapter 1, and the min-sum algorithm [KFL01].[7]

There can be wide variability in the message-passing schedule, and the choice of schedule can determine the computational efficiency of the algorithm. In one schedule, each node updates its outgoing messages whenever it receives a subset consisting of K incoming messages (if it has a total of $K+1$ edges). Then whenever any incoming messages get updated, the outgoing message also get updated. Thus the nodes attached to external edges of the module are updated first, followed by the nodes connected to them. Another possible schedule is the "flooding schedule," in which messages are passed along all edges at all times simultaneously. When the graph consists of nodes arranged in a linear fashion, there is a natural schedule, in which a decoder uses a single forward sweep and a single backward sweep of message-passing. This "forward-backward" algorithm is discussed in more detail in Chapter 3. For graphs without cycles, it turns out that if the stopping criterion is such that the decoder passes messages until there are no more updates

[7] As a variation on the message-passing algorithm, one might consider the economy as a distributed computation involving the exchange of numerical tokens between entities, an activity which might more accurately be termed the *money-passing algorithm*.

to be made, then regardless of the choice of message-passing schedule, the resulting outputs along the external edges are exactly the extrinsic probabilities with respect to the entire graph $\mathcal{G}$, as specified in (2.45). The computationally difficult problem of evaluating the extrinsic probabilities has been broken down into simple sub-computations based on local constraints at each of the nodes in the graph.

2.4.2 GRAPHS WITH CYCLES

If a graph contains cycles (also known as "loopy" graphs), which is the case in most situations of interest, the presence of cycles invalidates the independence assumptions in the message-passing algorithm. The situation of graphs with a single cycle has been analyzed (e.g., in Aji *et al.* [AHM98] and Weiss [Wei00]), but in general, when there are cycles, the message-passing algorithm serves only as an approximate algorithm without a guarantee on convergence.

One difference with the case of cycle-free graphs is that for graphs with cycles, it is necessary to have initial messages (in both directions) for the internal edges. A common setting for the initial value of the messages is the uniform distribution $\mu_{N_j \to N_{j'}}(x_i = a) = \frac{1}{|A_i|}$. If the internal edges do not start with messages, then the presence of cycles could cause the message-passing algorithm to stall, as no node is receiving enough messages to perform a computation. This is evident from the example of three nodes in Figure 2.9, where in the graph on the left side, the single incoming message leads to no further computations, since that node only has 1 incoming message but 3 edges. In the graph on the right side of Figure 2.9, the internal edges start off with default message, so that the message-passing algorithm can proceed as usual. Note that for graphs with cycles, the choice of message-passing schedule and stopping criterion can have an impact on the decoding complexity and performance.

Fortunately, it has been found in practice that for many situations of graphs with cycles, such in the cases of turbo and LDPC codes, the message-passing algorithm produces accurate estimates of the desired probabilities with much lower complexity than an exact decoding. While short cycles (e.g., of length 4) clearly cause the independence assumptions to be violated, if the cycles can be made large enough, then their effect becomes less detrimental. Note that when cycles are present, the MPA does not actually calculate the true extrinsic probabilities as in (2.45), but essentially solves a different problem whose answer is an approximation of the desired probabilities in (2.45). By exchanging messages between multiple simple decoders, the performance of a more complicated decoder can be approached.

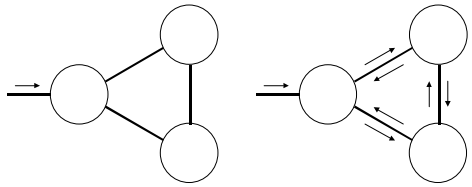

Figure 2.9. Initial message vectors are necessary for the internal edges in graphs with cycles

2.4.3 CONTINUOUS ALPHABETS

Note that the above discussion has been in terms of discrete alphabets A. It is also possible to use continuous-valued alphabets, such as the real numbers $A = \mathbb{R}$. Suppose that a variable $\{x_i\}$ belongs to the continous-valued alphabet $A = \mathbb{R}$. Then the message $\mu_{X_i \to N}(x_i = \xi)$ represents a *probability density function* over $\xi \in \mathbb{R}$ (rather than a probability mass function). As a result, it should actually be represented as a function $f(x_i = \xi_i)$ rather than by $P(x_i = \xi_i)$ (which implies a probability), since strictly speaking, the probability that x_i equals any particular value is equal to 0. As an abuse of notation, we will maintain consistency and continue to use the notation $P(x_i = \xi_i)$ to represent the probability density function in the case of a continuous alphabets. (Note that since message vectors are assumed to be equivalent up to multiplication by a constant, it is the relative proportion that is important anyways.)

Note that in the situations that we consider, even if the variables come from a continuous alphabet, the constraint sets S_N are all discrete sets. As a result, for the sum-product algorithm, it is not necessary to replace the summation by an integral (2.47).

2.4.4 APPLICATION

Broadly stated, the message-passing algorithm can be applied to any problem of the following type: Consider a set $\widehat{E}$ of variables, where each variable x_i belongs to its respective alphabet A_i, for $i = 1, 2, ..., \left|\widehat{E}\right|$. Suppose that a constraint set $\widehat{S}$ defines the valid configurations of the

values for these variables, where

$$\widehat{S} \subset A_1 \times A_2 \times \cdots \times A_{\left|\widehat{E}\right|}.$$

Let $P^{\text{int}}(x_i)$ be the intrinsic probability for x_i, and suppose that the variables are independent so that the joint probability factors into the product of the individual probabilities. Then the posterior probability taking into account this constraint $\widehat{S}$ turns out be equal to the sum of the product of intrinsic probabilities.

$$P^{\text{post}}(x_i) = c_i \sum_{\substack{\left(x_1,\ldots,x_{\left|\widehat{E}\right|}\right)\in\widehat{S} \\ \sim\{x_i\}}} \prod_l P^{\text{int}}(x_l), \tag{2.48}$$

The summation is over all configurations of values in the set $\widehat{S}$.

This framework can be used to describe many decoding problems, where the goal is to compute a posterior probability based on intrinsic probabilities, as defined by (2.48). In the case of error-control codes (ECCs), the variables in $\widehat{E}$ represent the codeword symbols, the intrinsic probabilities are derived from the received signal, the set $\widehat{S}$ represents the set of valid codewords, and the posterior probabilities represent the decoder output, which can be used to make bit decisions.

As this expression (2.48) can often be complicated to evaluate, the idea is to introduce a graph $\mathcal{G}$ to express the relationships expressed by the constraint set $\widehat{S}$ on the variables in $\widehat{E}$ in order to simplify the computation. The graph $\mathcal{G}$ is chosen so that its external edge-variables correspond exactly to the variables in $\widehat{E}$, known as "symbol variables." In addition, the graph includes a set of nodes describing local constraints, as well as internal edge-variables, known as "state variables."

Based on this set of internal and external edge-variables $E_{\mathcal{G}}$ and the set of nodes $N_{\mathcal{G}}$, it is possible to define an overall constraint set $S_{\mathcal{G}}$, which should match the original constraint set $\widehat{S}$ when restricted to the external edge-variables. (Actually, the graph $\mathcal{G}$ is often chosen first, and then the set $\widehat{S}$ of valid codewords is defined from the constraint set $S_{\mathcal{G}}$ by looking at the external edge-variables.) With the graph representation $\mathcal{G}$, the solution of (2.48) can then be found by applying the message-passing algorithm.

3. NODES AND MODULES

Having set up the framework for soft iterative decoding, we discuss two basic types of nodes, the equality node and the parity-check node, which form the basic building blocks for low-density parity check (LDPC)

codes. Nodes are then organized into modules, and some key examples are introduced.

3.1 EQUALITY NODE

In the *equality node*, pictured in Figure 2.10, the constraint specifies the variables to be equal. (In the case of binary LDPC codes, the equality node is also known as a "bit node," since it corresponds to a single bit.) Suppose that the node N is connected to $K+1$ edges, with associated variables $x_0, x_1, ..., x_K$ that belong to the same alphabet A. Then the constraint set for this node is

$$S_N = \{(x_0, x_1, ..., x_K) \mid x_0 = x_1 = \cdots = x_K\},$$

and contains $|A|$ elements. Suppose that the input message vectors are $\mu_{X_i \to N}(x_i)$ for $i = 0, 1, ..., K$. Without loss of generality, let us consider the output along the edge-variable x_0. The update equation follows from (2.47),

$$\begin{aligned}\mu_{N\to X_0}(x_0 = \xi) &= c_{x_0} \sum_{\substack{(x_0,x_1,\ldots,x_K)\in S_N \\ \sim\{x_0\}}} \prod_{l=1}^{K} \mu_{X_l\to N}(x_l) \\ &= c_{x_0} \prod_{l=1}^{K} \mu_{X_l\to N}(x_l = \xi)\end{aligned} \tag{2.49}$$

where c_{x_0} is the normalization factor, which is equal to

$$c_{x_0} = \left(\sum_{\zeta\in A} \prod_{l=1}^{K} \mu_{X_l\to N}(x_l = \zeta) \right)^{-1}.$$

The summation is over all elements of the alphabet A for x_0. The sum of products reduces to a product of the intrinsic probabilities corresponding to the same value. This equation (2.49) can be applied similarly to compute the output along any edge-variable x_i, based on the inputs along the remaining edge-variables x_l for $0 \le l \le K$, and $l \ne i$.

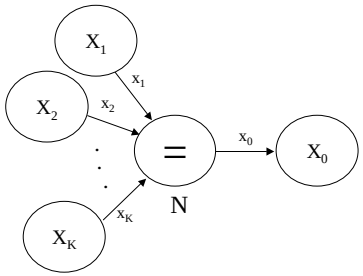

Figure 2.10. Message-passing at an equality node

In the case of a binary alphabet A, we can rewrite (2.49) using the log-likelihood ratio (LLR).

$$\begin{aligned}\mathrm{LLR}_{N\to X_0}(x_0) &= \log\frac{\mu_{N\to X_0}(x_0=1)}{\mu_{N\to X_0}(x_0=0)}\\ &= \log\frac{\prod_{i=1}^{K}\mu_{X_i\to N}(x_i=1)}{\prod_{i=1}^{K}\mu_{X_i\to N}(x_i=0)}\\ &= \sum_{i=1}^{K}\mathrm{LLR}_{X_i\to N}(x_i) \qquad (2.50)\end{aligned}$$

For an equality node, the outgoing message along one edge is the sum of the incoming messages along all of the other edges in terms of log-likelihood ratios. (Observe the similarity of equation (2.50) to the example of soldier-counting given in Chapter 1, where the output message is a sum of the incoming messages.)

The input and output for an equality node are summarized by equation (2.49) for probabilities and (2.50) for log-likelihood ratios. Let us consider some examples using binary variables, which are depicted in Figure 2.11:

- **Equality node with two edges.** Consider an equality node N on two binary variables, x_1, x_2. This corresponds to a repetition code, where the constraint set $S = \{(0,0),(1,1)\}$. Suppose that the inputs

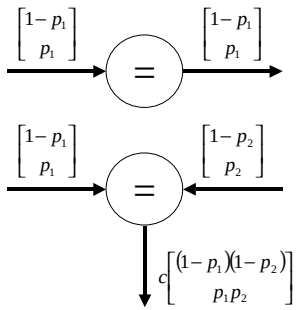

Figure 2.11. Equality nodes with two and three edges

to the nodes are denoted by

$$\vec{\mu}_{X_1\to N}(x_1) = \begin{bmatrix} 1-p_1 \\ p_1 \end{bmatrix}$$
$$\vec{\mu}_{X_2\to N}(x_2) = \begin{bmatrix} 1-p_2 \\ p_2 \end{bmatrix},$$

representing intrinsic probabilities with respect to the node N. Then following (2.49), the outputs from the node are given by

$$\mu_{N\to X_1}(x_1) = \mu_{X_2\to N}(x_2)$$
$$\mu_{N\to X_2}(x_2) = \mu_{X_1\to N}(x_1).$$

The posterior probability for x_i is then given as follows,

$$\begin{aligned}\vec{P}^{\,\mathrm{post}}(x_1) &= c\cdot(\vec{\mu}_{N\to X_1}(x_1)\odot\vec{\mu}_{X_1\to N}(x_1))\\ &= c\begin{bmatrix}(1-p_1)(1-p_2)\\ p_1p_2\end{bmatrix}, \qquad (2.51)\end{aligned}$$

where the normalization constant is given by

$$(c)^{-1} = p_1p_2 + (1-p_1)(1-p_2).$$

An identical expression to (2.51) holds for the posterior probability for x_2.

- **Equality node with three edges.** Next, consider an equality node N with three binary variables, x_0, x_1, and x_2. Suppose the intrinsic probabilities on x_1 and x_2 are given by p_1 and p_2, respectively. Then the output extrinsic probability vector with respect to N is given by

$$\overrightarrow{P}^{\text{ext}}(x_0) = c \begin{bmatrix} (1-p_1)(1-p_2) \\ p_1 p_2 \end{bmatrix},$$

which is the same as the posterior probability for the case of an equality node with two edges in (2.51). In terms of log-likelihood ratios, we have

$$\text{LLR}_{N \to X_0}(x_0) = \text{LLR}_{X_1 \to N}(x_1) + \text{LLR}_{X_2 \to N}(x_2)$$
$$\log \frac{p_1 p_2}{(1-p_1)(1-p_2)} = \log\left(\frac{p_1}{1-p_1}\right) + \log\left(\frac{p_2}{1-p_2}\right).$$

3.2 PARITY-CHECK NODE

In a *parity-check node* (or "check node"), the local constraint specifies that the variables sum to 0. Consider a parity-check node N where the neighboring edges have associated variables $x_0, x_1, ..., x_K$ that belong to the same alphabet A, as shown in Figure 2.12. Suppose that the alphabet A is an additive group under the operation $\oplus$. Then the constraint set S_N is made up of all configurations that satisfy an additive constraint.

$$S_N = \{(x_0, x_1, ..., x_K) \mid x_0 \oplus x_1 \oplus \cdots \oplus x_K = 0\}$$

The update equation for x_0 is given by (2.47),

$$\mu_{N \to X_0}(x_0) = c_{x_i} \sum_{\substack{(x_0, x_1, ..., x_K) \in S_N \\ \sim \{x_0\}}} \prod_{l=1}^{K} \mu_{X_l \to N}(x_l), \qquad (2.52)$$

which involves a summation of all configurations in S_N that are consistent with an implied value of $x_0 = \xi_0$, following the simplified notation.

In the case of binary variables, the alphabet is $A = \mathbb{Z}_2$, the additive group of two elements. For a parity-check node, the constraint set S_N consists of 2^K configurations for the $K+1$ variables, since the constraint eliminates exactly half of the possible configurations. Then in (2.52), the summation is over all elements of S_N that are consistent with a fixed value for x_i. This summation over 2^{K-1} elements appears complicated, but fortunately, it turns out that there are ways to simplify this computation.

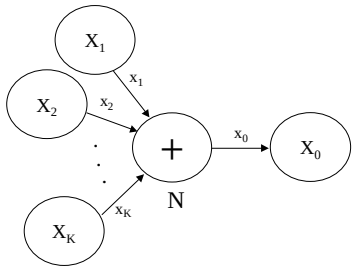

Figure 2.12. Message-passing at a parity-check node

With respect to the parity check node N, the input messages correspond to the intrinsic probabilities $p_i = P^{\text{int}}(x_i = 1)$.

$$\mu_{X_i \to N}(x_i = 0) = 1 - p_i$$
$$\mu_{X_i \to N}(x_i = 1) = p_i$$

The output message along edge-variable x_0 should be the extrinsic probability with respect to N. This is given by the following probabilities:

$$\mu_{N \to X_0}(x_0 = 0) = P(x_1 \oplus \cdots \oplus x_K = 0)$$
$$\mu_{N \to X_0}(x_0 = 1) = P(x_1 \oplus \cdots \oplus x_K = 1)$$

To evaluate these expressions, we first consider the case of three edges (so $K = 2$). In this case, $x_1 \oplus x_2$ equals 0 exactly when both variables have the same value, so that

$$\begin{aligned} \mu_{N \to X_0}(x_0 = 0) &= P(x_1 \oplus x_2 = 0) \\ &= p_1 p_2 + (1-p_1)(1-p_2) \end{aligned} \qquad (2.53)$$

and

$$\mu_{N \to X_0}(x_0 = 1) = p_1(1-p_2) + (1-p_1)p_2,$$

as shown in Figure 2.13.

This expression (2.53) can be rewritten as

$$2P(x_1 \oplus x_2 = 0) - 1 = (1-2p_1)(1-2p_2), \qquad (2.54)$$

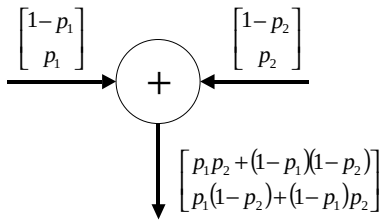

Figure 2.13. Example of a parity-check node

which can be generalized for $K > 2$ as follows. Given the following definitions,

$$\begin{aligned} M_K &= P\left(x_1 \oplus x_2 \oplus \cdots \oplus x_K = 0\right) \\ M_{K-1} &= P\left(x_1 \oplus x_2 \oplus \cdots \oplus x_{K-1} = 0\right) \\ p_K &= P^{\text{int}}\left(x_K = 1\right) \end{aligned}$$

suppose by induction that the expression (2.54) is true for the case of $K-1$ variables, so that

$$2M_{K-1} - 1 = \prod_{i=1}^{K-1} \left(1 - 2p_i\right).$$

Then the expression $x_1 \oplus x_2 \oplus \cdots \oplus x_K$ can be obtained as the sum of x_K and $x_1 \oplus x_2 \oplus \cdots \oplus x_{K-1}$. Following a calculation similar to (2.53), we obtain

$$\begin{aligned} M_K &= \left(1 - p_K\right) M_{K-1} + p_K \left(1 - M_{K-1}\right) \\ &= \left(1 - 2p_K\right) M_{K-1} + p_K, \end{aligned}$$

which can be rewritten as

$$\begin{aligned} 2M_K - 1 &= 2\left(1 - 2p_K\right) M_{K-1} + 2p_K - 1 \\ &= \left(1 - 2p_K\right)\left(2M_{K-1} - 1\right), \end{aligned}$$

thus yielding the desired result,

$$2M_K - 1 = \prod_{i=1}^{K} \left(1 - 2p_i\right). \tag{2.55}$$

By the induction hypothesis, this expression (2.55) is valid for all $K \geq 1$. Solving for M_K, we obtain

$$M_K = \frac{1 + \prod_{i=1}^{K} \left(1 - 2p_i\right)}{2},$$

so that the output messages can be written in terms of the input messages $p_i = \mu_{X_i \to N}\left(x_i = 1\right)$ as follows, giving a simplification of the update equation (2.52).

$$\mu_{N \to X_0}\left(x_0 = 0\right) = \frac{1 + \prod_{i=1}^{K} \left(1 - 2p_i\right)}{2} \tag{2.56}$$

$$\mu_{N \to X_0}\left(x_0 = 1\right) = \frac{1 - \prod_{i=1}^{K} \left(1 - 2p_i\right)}{2} \tag{2.57}$$

These equations (2.56) and (2.57) can also be found using a Fourier transform approach that can be generalized to non-binary alphabets, as shown in Appendix B.

Since the variables are binary, the expressions (2.56) and (2.57) can be combined using the log-likelihood ratio (LLR).

$$\begin{aligned} \text{LLR}_{N \to X_0}\left(x_0\right) &= \log \frac{\mu_{N \to X_0}\left(x_0 = 1\right)}{\mu_{N \to X_0}\left(x_0 = 0\right)} \\ &= \log \frac{1 - \prod_{i=1}^{K} \left(1 - 2p_i\right)}{1 + \prod_{i=1}^{K} \left(1 - 2p_i\right)} \end{aligned} \tag{2.58}$$

The hyperbolic tangent function and its inverse,

$$\begin{aligned} \tanh\left(\frac{x}{2}\right) &= \frac{e^x - 1}{e^x + 1} \\ 2\tanh^{-1}\left(y\right) &= \log \frac{1 - y}{1 + y}, \end{aligned}$$

can be applied to simplify this expression. First, the relation of the soft bit and LLR (cf. (2.13)) gives the following:

$$1 - 2p_i = -\tanh\left(\frac{1}{2}\text{LLR}\left(p_i\right)\right)$$

Then the outgoing message from this check node (2.58) can be cast purely in terms of log-likelihood ratios,

$$\begin{aligned}\mathrm{LLR}_{N\to X_0}(x_0) &= \log\frac{1-(-1)^K\prod_{i=1}^K\tanh\left(\frac{1}{2}\mathrm{LLR}(p_i)\right)}{1+(-1)^K\prod_{i=1}^K\tanh\left(\frac{1}{2}\mathrm{LLR}(p_i)\right)}\\ &= (-1)^{K-1}\,2\tanh^{-1}\left(\prod_{i=1}^K\tanh\left(\frac{1}{2}\mathrm{LLR}_{X_i\to N}(x_i)\right)\right),\end{aligned} \tag{2.59}$$

where the fact that the inverse hyperbolic tangent $\tanh^{-1}(\cdot)$ is an odd function is used to move the sign $(-1)^{K-1}$ outside the function.

We remark that if we had defined the log-likelihood ratio with the opposite convention (i.e. $\log\frac{P(x=0)}{P(x=1)}$) as in [Gal63], then the factor of $(-1)^{K-1}$ disappears in (2.59). We have chosen our LLR definition in order to have a positive LLR correspond to $x=1$ and negative LLR to $x=0$.

3.3 MODULES

Having defined several nodes, we can organize them into modules. As with nodes, each module is a "soft-in, soft-out" decoder, where for every external edge of the module, the input is the intrinsic probability and the output is the extrinsic probability with respect to the module. Since the module can be used a part of a larger system, the module is also known as a "graph fragment" in [For01]. In other words, since the entire system can be represented by an overall normal graph, a module is simply a way to specify a fragment of that larger graph that represents a particular function. As described in Section 3.3.1, it is possible to connect modules together using multiple edges.

This use of modules coincides very naturally with the usual block diagram notation for communication systems, where the edges between blocks represent a stream of signals or bits, as in Chapter 1. These block diagrams can then be interpreted using the soft iterative decoding framework, where each block is a module and signals are replaced by messages corresponding to probability distributions. With this representation, the block diagrams for the encoder and decoder are often very similar, and in many cases are identical—the only difference is that the encoder starts with messages corresponding to the known user data at one end of the block diagram, while decoder starts with messages corresponding to a noisy received signal at the other end of the block diagram.

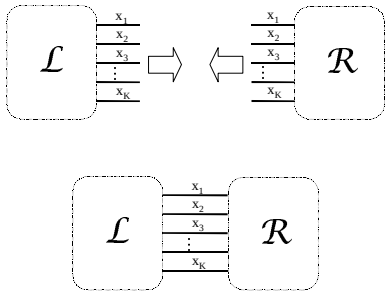

Figure 2.14. Connecting two modules, in which two bundles of external edges combine to form one bundle of internal edges.

3.3.1 MULTIPLE EDGES BETWEEN MODULES

It is possible to have multiple edges connecting two modules, as shown in Figure 2.14. A set of parallel edges is referred to as a *bundle* of edges. The top part of the figure shows two modules, each with a bundle of external edges. These can be combined to form a single bundle of edges, as shown on the bottom. In the graph composed of the combination of these two modules, the two bundles of external edges have now become a single bundle of internal edges. As a notational convenience, we can also represent the bundle by a thick line, as shown in Figure 2.15. (Alternatively, the bundle can be represented by a line with a slash through it, in accordance with the notation for wires in a circuit diagram.)

It should be noted that for a bundle of edges, the message is handled separately for each edge in the bundle. As shown in Figure 2.16, just as for a node, the output along each edge of a module is a function of the inputs along all the other edges, including the other edges in the same bundle. Suppose that the bundle consists of K edges that are associated with binary variables x_1, x_2, ..., x_K (where the alphabet is $A=\{0,1\}$). Then the bundle carries a separate message vector for edge x_i, so that the message for the bundle is actually a collection of K separate vectors:

$$\left(\begin{bmatrix}1-p_1\\ p_1\end{bmatrix},\begin{bmatrix}1-p_2\\ p_2\end{bmatrix},\ldots,\begin{bmatrix}1-p_K\\ p_K\end{bmatrix}\right)$$

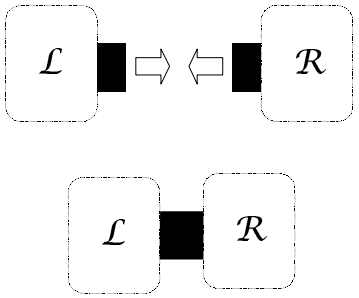

Figure 2.15. Connecting two modules, as shown using a thick line to represent the a bundle of edges

There is 1 degree of freedom in each edge and thus a total of K degrees of freedom. Note that using bundles of edges makes it likely that the graph contains cycles, so that methods should be used (e.g., the permuter module in Section 3.3.3) to increase the length of the cycles so that their effect is mitigated.

On the other hand, an alternative is to treat all the binary variables in the bundle as a single variable, in which case the joint alphabet would be $A \times A \times \cdots \times A = \{0,1\}^K$ and the message vector would be of length 2^K, having $2^K - 1$ degrees of freedom, rather than K degrees of freedom. This approach avoids the introduction of cycles into the graph, but for most decoding applications, this level of detail is excessive and unwieldy.

3.3.2 EQUALITY MODULE

Recall that an equality node is used for expressing an equality relation between edge-variables. Equality nodes can be combined together into an equality module that serves the purpose of expressing an equality relation between the corresponding edges of multiple bundles.

For an equality module with three bundles of edges, as shown in Figure 2.17, suppose the input messages from the left and top are

$$\left(\overrightarrow{\mu}\left(x_1\right), \overrightarrow{\mu}\left(x_2\right), ..., \overrightarrow{\mu}\left(x_K\right)\right)$$

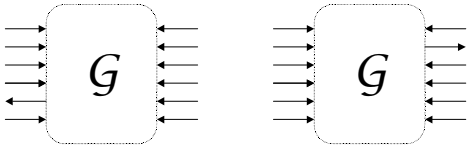

Figure 2.16. The output on an edge depends on the inputs on all other edges

and

$$\left(\overrightarrow{\mu}\left(x_1'\right), \overrightarrow{\mu}\left(x_2'\right), ..., \overrightarrow{\mu}\left(x_K'\right)\right),$$

respectively, where each $\overrightarrow{\mu}\left(x_1\right)$ is a message vector of the form

$$\overrightarrow{\mu}\left(x_i\right) = \begin{bmatrix} \mu\left(x_i = a_1\right) \\ \mu\left(x_i = a_2\right) \\ \vdots \\ \mu\left(x_i = a_{|A|}\right) \end{bmatrix}$$

All edge-variables $\{x_i\}$ and $\{x_i'\}$ are assumed to the same alphabet A. Then the output message on the right is given by the following collection of message vectors

$$\left(c_1 \overrightarrow{\mu}\left(x_1\right) \odot \overrightarrow{\mu}\left(x_1'\right), c_2 \overrightarrow{\mu}\left(x_2\right) \odot \overrightarrow{\mu}\left(x_2'\right), ..., c_K \overrightarrow{\mu}\left(x_K\right) \odot \overrightarrow{\mu}\left(x_K'\right)\right),$$

where $\odot$ represents the pointwise product of the two message vectors, and c_i is the normalization constant so that each of the resulting K message vectors sum to 1.

3.3.3 PERMUTER MODULE

The permuter module is often called a random "interleaver" in turbo code design. When a bundle of edges exists between two modules, the system designer needs to be careful not to introduce short cycles into the overall graph of the two modules. For example, directly connecting two modules where there is a relation between adjacent elements (e.g., x_i and x_{i+1}) could lead to short cycles. As a result, some randomization of the order may be necessary, which can be accomplished through a permuter module, also known as an "interleaver" in turbo coding.

As shown in Figure 2.17, there are two bundles of edges to this module. This module then permutes the set of messages between the two bundles.

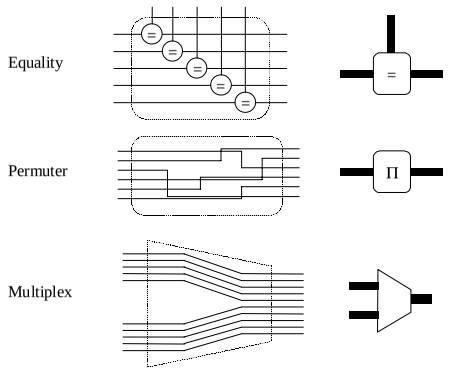

Figure 2.17. Modules

Let $\overrightarrow{\mu}(x_i)$ denote a message vector for the variable x_i on the left bundle. If the input messages from the left are

$$(\overrightarrow{\mu}(x_1), \overrightarrow{\mu}(x_2), ..., \overrightarrow{\mu}(x_K)),$$

then the output messages on the right are

$$(\overrightarrow{\mu}(x_{\pi(1)}), \overrightarrow{\mu}(x_{\pi(2)}), ..., \overrightarrow{\mu}(x_{\pi(K)})),$$

where π represents a permutation of the set $\{1, 2, ..., K\}$. Note that a set of message vectors passed in the reverse direction will experience an inverse permutation (denoted π^{-1}) from the message vectors in the forward direction. There are no nodes involved in this permuter module, and no computation is required. This permuter can use a permutation π that is tailored for a particular situation; in some applications, with high probability, a randomly selected permutation suffices.

3.3.4 MULTIPLEX MODULE

The *multiplex module* combines two (or more) bundles of edges into a single bundle, and in reverse, it takes a bundle and splits it into two (or more) bundles. As shown in Figure 2.17, if there are two bundles on

the left of the multiplex module consisting of

$$(\overrightarrow{\mu}(x_1), \overrightarrow{\mu}(x_2), ..., \overrightarrow{\mu}(x_K)),$$

and

$$(\overrightarrow{\mu}(x'_1), \overrightarrow{\mu}(x'_2), ..., \overrightarrow{\mu}(x'_{K'})),$$

then the message vectors for the combined bundle on the right are

$$(\overrightarrow{\mu}(x_1), \overrightarrow{\mu}(x_2), ..., \overrightarrow{\mu}(x_K), \overrightarrow{\mu}(x'_1), \overrightarrow{\mu}(x'_2), ..., \overrightarrow{\mu}(x'_{K'})).$$

In the reverse direction, this multiplex module splits the combined bundle of edges into two smaller bundles.

3.3.5 PROBABILISTIC CIRCUITS

An analogy can be made between the normal graph and a digital circuit. The nodes correspond to the basic logic gates, and the edges correspond to the wires of this system. The exchange of probabilistic messages along the edges of the graph corresponds to the flow of electric current through the wires in a circuit. A module is a collection of nodes and edges that may be used as a component in a system, in analogy to the way an integrated circuit, composed of transistors and wires, can be used as a component on a board.[8]

Table 2.2 suggests some analogous features between digital circuits and these normal graphs, which we might think of as "probabilistic circuits." In digital logic, the messages sent on the wires are assumed to be binary (with high voltage for 1 and low voltage for 0), while in a graph, the messages are vectors of probabilities. Instead of logic gates, the message-passing algorithm has nodes that operate on probabilities. The external edges act as the input/output (I/O) interface for the module, while a bundle of edges acts like a data bus carrying a parallel set of signals. Finally, the parity-check node corresponds to an XOR gate, since it expresses a relation where the involved variables sum to 0, and the equality node corresponds to a register (e.g., for storing a single bit), since the constraint set specifies that the edge-variables should be equal, whereas all wires connected to a register have the same value.

One difference, however, is that in a digital circuit, the current flows in one direction along the wire, whereas the probabilistic messages flow in both directions along the edges of the graph. (Note that the implementation of message-passing decoders using analog circuits takes this analogy between circuits and graphs one step further. [LLHT01].)

[8] These days, a more appropriate analogy might be an IP (intellectual property) core that is integrated into an ASIC design.

digital circuit	normal graph
gate	node (function)
wire	edge (variable)
signal	message vector (probabilities)
integrated circuit (IC)	module (graph fragment)
internal wire	internal edge
I/O interface	external edge
data bus	bundle of edges
XOR gate	parity-check node
register	equality node

Table 2.2. Analogy between circuits and normal graphs

4. LOW-DENSITY PARITY-CHECK CODES

Having presented the framework of normal graphs and the message-passing algorithm, it is straightforward to discuss low-density parity-check (LDPC) codes and their decoding algorithm. LDPC codes (also known as Gallager codes) are error-correcting codes with sparse parity-check matrices, which are amenable to decoding using the message-passing algorithm. They were introduced by Gallager [Gal62][Gal63] in the early 1960's, revisited by Tanner [Tan81], and only after the advent of turbo codes, were they rediscovered by MacKay and Neal [MN97] [Mac99], and Sipser and Spielman [SS96].

4.1 DEFINITION

A binary *low-density parity check (LDPC)* code is a binary linear error-correcting code defined by a sparse parity-check matrix H. If the parity-check matrix H has N columns and M rows, the codewords consist of sequences v of N bits that satisfy a set of M binary parity checks defined by the parity-check equation $Hv = 0$, as shown in Figure 2.18. The number of message bits is $K = N - M$, and the rate of the code is $\frac{K}{N}$, assuming that the matrix is of full rank. (See Appendix B for the extension of this definition to non-binary alphabets.)

A straightforward method for generating the set of valid codewords is to reduce H to systematic form $H_{sys} = [I_M \mid P]$ by performing binary Gaussian elimination and reordering the columns appropriately. (It will be assumed that H has full rank. If some rows of H are linearly dependent, remove the extra rows, and let M equal the resulting number of independent rows.) For H in systematic form, the systematic bits u form a subset of the codeword bits v. (In most cases, the systematic bits are used as the user bits.) A corresponding generator matrix is

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

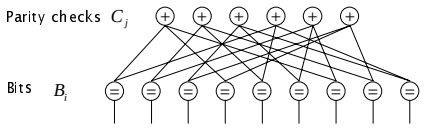

Figure 2.18. Parity-check matrix and its associated graph

$G_{sys} = \left[-P^T \mid I_{N-M}\right]$. Since the matrix P is likely to be uniformly dense (with half 1's and half 0's), the denseness of P makes encoding using G_{sys} via matrix multiplication high in complexity. Methods have been found for greatly reducing the complexity of the encoding process (ref. Richardson and Urbanke [RU01b]). Using codes that have structure (as in Chapter 8) is another way to reduce the complexity of the encoding process. Another approach used by Cheng [Che97] is low-density generator matrix (LDGM) codes, in which H is chosen to be both systematic and sparse. These are easily encoded, but in general do not perform as well as matrices that are not restricted to be systematic.

The graph consists of "bit nodes," which are equality nodes (from Section 3.1) that represent the bits of the codeword, and "check nodes" (from Section 3.2) that express the parity-check relationships. This parity check matrix H provides the structure for the decoding algorithm. As shown in Figure 2.18, there is an edge in the graph connecting the bit and check nodes exactly when there is a "1" in the parity check matrix, so that an association can be made between edge-variables and the non-zero entries of the parity check matrix. The edge-variables in the same row of the parity-check matrix H are connected to the same check node,

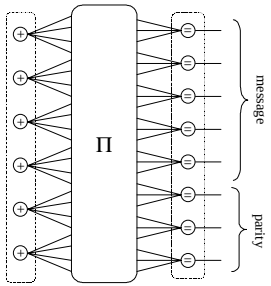

Figure 2.19. Module-based representation for LDPC code

and so satisfy a parity-check constraint. Meanwhile, the edge-variables in the same column of the parity-check matrix H are connected to the same bit node, and so satisfy an equality constraint. The graph for an LDPC code can also be represented using a permuter module Π to describe the connections between the bits and checks, as shown in Figure 2.19 (ref. [For01]). In other words, the details of how the edges connect together are absorbed by the permuter module.

This graph representation completely describes all the relations of the code, and can be used for decoding using the message-passing algorithm. Starting with the input consisting of intrinsic probabilities for the bits, the message-passing algorithm uses the parity-check relationships amongst the bits to iteratively pass messages between the bit nodes and check nodes to obtain extrinsic probabilities for the bits. The intrinsic and extrinsic probabilities can then be combined to give posterior probabilities for the codeword bits that incorporate the knowledge gained from the LDPC code. Despite the fact that the graphs associated to LDPC codes may possess many cycles, for randomly chosen sparse matrices with large block lengths, it turns the algorithm computes accurate estimates of the extrinsic and posterior probabilities. See Richardson and Urbanke [RU01a] and Chung *et al.* [CRU01] for analyses of the capacity of LDPC codes.

In terms of the design of low-density parity-check codes, it was shown by Gallager [Gal63] that for large block lengths, the minimum distance of the code grows linearly with blocklength, so that a randomly chosen LDPC code will have large distance with high probability, making it a

powerful error-correcting code. Hence, the first design principle is to use a very long block length if possible, so that with high probability, choosing a random code will give good performance. Note that there can be many equivalent parity-check matrices for the same code, and it is preferable to use a parity check matrix with low density so as to avoid short cycles in the graph (which violate the independence assumptions of the message-passing algorithm). In particular, cycles of length 4, in which two bits are both connected by edges to the same two checks, should especially be avoided.

The choice of the degrees of these nodes (corresponding to the number of ones in a column or row of the parity check matrix) determines the performance of the LDPC decoder. Codes with the same degree at each node are known as "regular" LDPC codes, with the others known as "irregular." It has been found that with careful choice of the distribution of degrees (ref. Richardson *et al.* [RSU01] and Luby *et al.* [LMSS01b]) and sufficiently long block lengths, it is possible to design irregular LDPC codes that allow for error-free transmission at rates that are extremely close to the Shannon capacity of the AWGN channel, performing even better than turbo codes.

While it has been found that certain irregular constructions perform better than others, for our simulations, we find it convenient to use matrices that are column-regular. These parity-check matrices are constructed by randomly selecting t_c locations in each column and putting a 1 in that location, and special care has been taken to avoid cycles of length 4.

4.2 LDPC DECODER USING PROBABILITIES

Using the update equations for the bit nodes (a.k.a. equality nodes) and check nodes (a.k.a. parity-check nodes) given above, we can perform decoding on the graph for an LDPC code, as shown in Figure 2.20. For a parity-check matrix H of size $M \times N$, suppose that the bit nodes are B_i for $i = 1, 2, ..., N$, and the check nodes are C_j for $j = 1, 2, ..., M$. If an edge exists between the nodes B_i and C_j, then that edge is labeled by the variable e_{ji}. These are the internal edges of the graph for the LDPC code. In addition, there is also an edge for each bit node B_i that is associated with the variable v_i. These edges correspond to the external edges of the graph for the LDPC code. In addition, it will be convenient to introduce a node N_i (which belongs to some other decoder or module) that is connected to B_i via the edge-variable v_i. With this node N_i, then the intrinsic probability w.r.t. the LDPC decoder can be denoted as the message $\mu_{N_i \to B_i}(v_i) = P^{\text{int}}_{\text{LDPC}}(v_i)$.

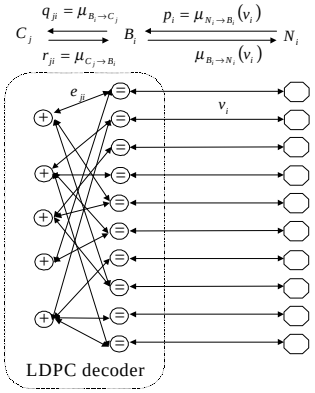

Figure 2.20. Graph for an LDPC decoder

The message-passing algorithm starts with a set of incoming probabilities $\mu_{N_i \to B_i}(v_i)$ corresponding to $P^{\text{int}}_{\text{LDPC}}(v_i)$. According to (2.49), the message from bit nodes B_i to check nodes C_j is given by the expressions,

$$\mu_{B_i \to C_j}(e_{ji} = 0) = c_{ji}\mu_{N_i \to B_i}(v_i = 0) \prod_{j' \in \mathsf{M}(i) \setminus \{j\}} \mu_{C_{j'} \to B_i}(e_{ji} = 0) \tag{2.60}$$

$$\mu_{B_i \to C_j}(e_{ji} = 1) = c_{ji}\mu_{N_i \to B_i}(v_i = 1) \prod_{j' \in \mathsf{M}(i) \setminus \{j\}} \mu_{C_{j'} \to B_i}(e_{ji} = 1) \tag{2.61}$$

where $\mathsf{M}(i)$ is the set of parity checks in which bit v_i is involved. This is also the set of row locations in the i-th column of the parity check matrix that contain a 1. The set $\mathsf{M}(i) \setminus \{j\}$ means the set $\mathsf{M}(i)$ with the element j omitted. (Note that the normalization constant c_{ji} is optional here since it can be restored at the end of the decoding procedure.)

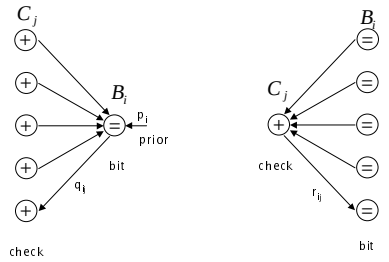

Figure 2.21. Message-passing from bit node to check node, and vice versa

On the other hand, according to (2.56) and (2.57), the messages from check node C_j to bit node B_i are given by the expressions,

$$\mu_{C_j \to B_i}(x_0 = 0) = \frac{1}{2}\left(1 + \prod_{i' \in \mathsf{L}(j) \setminus \{i\}} \left(1 - 2\mu_{B_{i'} \to C_j}(x_{i'} = 1)\right)\right) \tag{2.62}$$

$$\mu_{C_j \to B_i}(x_0 = 1) = \frac{1}{2}\left(1 - \prod_{i' \in \mathsf{L}(j) \setminus \{i\}} \left(1 - 2\mu_{B_{i'} \to C_j}(x_{i'} = 1)\right)\right) \tag{2.63}$$

where $\mathsf{L}(j)$ is the set of bit nodes connected to the j-th parity check, or the set of columns in the j-th row that contain a 1.

These two pairs of equations (2.60, 2.61) and (2.62, 2.63) describe the message-passing algorithm for the low-density parity-check code. This representation can be related to the notation used elsewhere (e.g., [Mac99]) by introducing the following variables:

$$\begin{aligned} p_i^b &= \mu_{N_i \to B_i}(v_i = b) \\ q_{ji}^b &= \mu_{B_i \to C_j}(e_{ji} = b) \\ r_{ji}^b &= \mu_{C_j \to B_i}(e_{ji} = b) \end{aligned}$$

for $b = 0, 1$, as shown in Figure 2.21. Then the update equations (2.60) and (2.61) for the messages from bit node B_i to the check node C_j may

be rewritten as

$$q_{ji}^0 = c'_{ji} \cdot p_i^0 \prod_{j' \in \mathsf{M}(i) \backslash \{j\}} r_{j'i}^0 \tag{2.64}$$

$$q_{ji}^1 = c'_{ji} \cdot p_i^1 \prod_{j' \in \mathsf{M}(i) \backslash \{j\}} r_{j'i}^1, \tag{2.65}$$

where c'_{ji} is a normalization constant. The update equations (2.62) and (2.63) for the message sent from check node C_j to bit node B_i may be rewritten as

$$r_{ji}^0 = \frac{1}{2}\left(1 + \prod_{i' \in \mathsf{L}(j) \backslash \{i\}} \delta q_{ji}\right) \tag{2.66}$$

$$r_{ji}^1 = \frac{1}{2}\left(1 - \prod_{i' \in \mathsf{L}(j) \backslash \{i\}} \delta q_{ji}\right), \tag{2.67}$$

where $\delta q_{ji} = q_{ji}^0 - q_{ji}^1 = 1 - 2q_{ji}^1$. The decoding algorithm starts with the intrinsic probabilities $\left(p_i^0, p_i^1\right)$, and uniform distributions for r_{ji} (i.e., $r_{ji}^0 = r_{ji}^1 = \frac{1}{2}$). Then the relations (2.64) and (2.65) yield the messages q_{ji} from bits to checks. (For the first iteration, the messages are $q_{ji}^b = p_i^b$.) The messages q_{ji} are then used in (2.66) and (2.67) to calculate the messages r_{ji} from checks to bits.

These two steps (updating q_{ji} and updating r_{ji}) comprise a single iteration of the message passing algorithm. The extrinsic probability w.r.t. the LDPC decoder $P_{\mathrm{LDPC}}^{\mathrm{ext}}(v_i)$ may be found by computing the outgoing message for each bit node v_i,

$$\mu_{B_i \to N_i}(v_i = 0) = c'_i \cdot \prod_{j \in \mathsf{M}(i)} r_{ji}^0$$

$$\mu_{B_i \to N_i}(v_i = 1) = c'_i \cdot \prod_{j \in \mathsf{M}(i)} r_{ji}^1,$$

where c'_i is a normalizing constant. Finally, the estimate of the posterior probability can be found by multiplying this extrinsic probability with the intrinsic probability as in (2.12),

$$q_i^b = c''_i \mu_{B_i \to N_i}(v_i = b)\, \mu_{N_i \to B_i}(v_i = b)$$

for $b = 0, 1$. In other words, the posterior probabilities are

$$q_i^0 = c_i \cdot p^0 \prod_{j \in \mathsf{M}(i)} r_{ji}^0$$

$$q_i^1 = c_i \cdot p^1 \prod_{j \in \mathsf{M}(i)} r_{ji}^1,$$

where the constant c_i is a normalization constant. It is hoped that after a number of iterations of the message-passing algorithm, these estimates of the posterior probabilities $\left(q_i^0, q_i^1\right)$ converge to the actual posterior probabilities w.r.t. the LDPC code.

4.3 LDPC DECODER USING LLRS

Since the variables are binary, these computations can also be described in terms of the log-likelihood ratios. The message $\mathrm{LLR}(q_{ji})$ from a bit node to a check node may be found using (2.50).

$$\mathrm{LLR}(q_{ji}) = \sum_{j' \in \mathsf{M}(i) \backslash \{j\}} \mathrm{LLR}\left(r_{j'i}\right) + \mathrm{LLR}(p_i) \tag{2.68}$$

Following (2.59), the message $\mathrm{LLR}(r_{ji})$ from a check node to a bit node can be written as

$$\mathrm{LLR}(r_{ji}) = (-1)^{|\mathsf{L}(j)|} 2 \tanh^{-1}\left(\prod_{i' \in \mathsf{L}(j) \backslash \{i\}} \tanh\left(\frac{1}{2}\mathrm{LLR}(q_{ji'})\right)\right). \tag{2.69}$$

(Again, if the opposite convention were chosen for the log-likelihood ratio, then the factor of $(-1)^{|\mathsf{L}(j)|}$ could be omitted.) The posterior LLR is given by summing over all the checks that contain the i-th bit.

$$\overbrace{\mathrm{LLR}(q_i)}^{\text{posterior}} = \overbrace{\sum_{j' \in \mathsf{M}(i)} \mathrm{LLR}\left(r_{j'i}\right)}^{\text{extrinsic}} + \overbrace{\mathrm{LLR}(p_i)}^{\text{intrinsic}} \tag{2.70}$$

With respect to the LDPC decoder, the intrinsic, extrinsic, and posterior LLRs satisfy the relation (2.14). Based on this posterior probability $\mathrm{LLR}(q_i)$, bit decisions can be made and a decoded word $\widehat{v}$ can be found. If this word satisfies the parity-check matrix ($H\widehat{v}^T = 0$), then it is a codeword, giving a natural stopping criterion, which reduces the average number of iterations.

To reduce the complexity for hardware implementation, it is desirable to transform the multiplications in equation (2.69) into additions. First,

observe that any product of real numbers can be written as follows,

$$\prod_i a_i = \left(\prod_i \mathrm{sgn}(a_i)\right) \exp\left(\sum_i \log(|a_i|)\right). \tag{2.71}$$

By introducing the function

$$\Psi(x) = -\log\left(\tanh(\frac{x}{2})\right) = \log\frac{1+\exp(-x)}{1-\exp(-x)},$$

which is defined for $x > 0$, the equation (2.69) can be decomposed into sign and amplitude using (2.71),

$$\mathrm{LLR}\left(r_{ji}\right) = (-1)^{|\mathsf{L}(j)|}\, s_{ji} \cdot \Psi\left(\sum_{i' \in \mathsf{L}(j)\setminus\{i\}} \Psi\left(\left|\mathrm{LLR}\left(q_{ji'}\right)\right|\right)\right) \tag{2.72}$$

where the sign is given by

$$s_{ji} = \prod_{i' \in \mathsf{L}(j)\setminus\{i\}} \mathrm{sgn}\left(\mathrm{LLR}\left(q_{ji'}\right)\right). \tag{2.73}$$

As observed by Gallager [Gal63], the function $\Psi(x)$ is equal to its own inverse $\Psi^{-1}(x)$ over the range $x > 0$. The function $\Psi(x)$ is depicted in Figure 2.22. Equation (2.72) thus provides an update equation using a single transformation function Ψ and some additions and binary operations (to compute the sign).

Note in (2.72), it is the large values that dominate in the summation. In fact, if there is a single large value $\Psi\left(\left|\mathrm{LLR}\left(q_{ji'}\right)\right|\right)$, the summation will be a large positive number, and the other terms do not matter. Since the function $\Psi(x)$ is monotonically decreasing for $x > 0$, a large value of $\Psi\left(\left|\mathrm{LLR}\left(q_{ji'}\right)\right|\right)$ corresponds to a small value of $\left|\mathrm{LLR}\left(q_{ji'}\right)\right|$. Hence, as an approximation, it is possible to replace the summation by the minimum value of $\left|\mathrm{LLR}\left(q_{ji'}\right)\right|$ over all i' in the relevant range ($i' \in \mathsf{L}(j)\setminus\{i\}$), using the fact that $\Psi(\Psi(x)) = x$. In other words, the expression (2.72) can be approximated by

$$\mathrm{LLR}\left(r_{ji}\right) \approx (-1)^{|\mathsf{L}(j)|}\, s_{ji} \cdot \min_{i' \in \mathsf{L}(j)\setminus\{i\}} \left(\left|\mathrm{LLR}\left(q_{ji'}\right)\right|\right), \tag{2.74}$$

where the sign s_{ji} is calculated as before in (2.73). This "*min*" *approximation* can yield a significant decrease in complexity with only a small degradation in performance, as shown in Fossorier *et al.* [FMI99].

Putting this all together, we summarize the decoding of low-density parity-check codes. The algorithm is given in a form suitable for implementation, i.e., using log-likelihood ratios and the function $\Psi(x)$ to convert multiplications into additions.

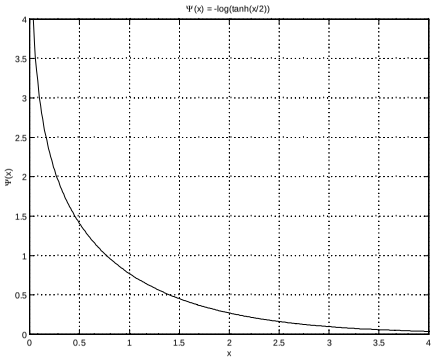

Figure 2.22. The function $\Psi(x)$ for converting products into sums in the LLR form of the LDPC decoding.

Decoding algorithm for LDPC codes using LLRs.

- Step 0. **Initialize.** The input to the decoding algorithm is a message vector (p_i^0, p_i^1) for each bit v_i, giving the intrinsic probability with respect to the decoder for each bit. As a log-likelihood ratio, this intrinsic information can be represented as

$$\begin{aligned}\mathrm{LLR}\left(p_i\right) &= \mathrm{LLR}^{\mathrm{int}}_{\mathrm{LDPC}}(v_i) \\ &= \log\frac{P^{\mathrm{int}}_{\mathrm{LDPC}}(v_i = 1)}{P^{\mathrm{int}}_{\mathrm{LDPC}}(v_i = 0)}.\end{aligned} \tag{2.75}$$

  In addition, the check-to-bit message vectors start off set to a uniform distribution, so that $\left(r_{ji}^0, r_{ji}^1\right) = \left(\frac{1}{2}, \frac{1}{2}\right)$, and

$$\mathrm{LLR}^{(0)}\left(r_{ji}\right) = \log\frac{r_{ji}^1}{r_{ji}^0} = 0.$$

  The iteration number k starts at 1.

- Step 1. **Bit-to-check messages.** The messages from bit nodes to check nodes are given by (2.68),

$$\mathrm{LLR}^{(k)}\left(q_{ji}\right) = \sum_{j' \in \mathsf{M}(i) \backslash \{j\}} \mathrm{LLR}^{(k-1)}\left(r_{j'i}\right) + \mathrm{LLR}\left(p_i\right).$$

- Step 2. **Check-to-bit messages.** The messages from check nodes to bit nodes are given by (2.72),

$$\mathrm{LLR}^{(k)}\left(r_{ji}\right) = (-1)^{|\mathsf{L}(j)|} \left(\prod_{i' \in \mathsf{L}(j) \backslash \{i\}} \mathrm{sgn}\left(\mathrm{LLR}^{(k)}\left(q_{ji'}\right)\right) \right) \cdot \Psi\left(\sum_{i' \in \mathsf{L}(j) \backslash \{i\}} \Psi\left(\left| \mathrm{LLR}^{(k)}\left(q_{ji'}\right) \right| \right) \right),$$

  where $\Psi(x) = -\log\left(\tanh\left(\frac{x}{2}\right)\right)$.

- Step 2′. **Check-to-bit messages ("min" approximation).** As an alternative to Step 2, the messages from check nodes to bit nodes can be approximated using the minimum function (2.74):

$$\mathrm{LLR}^{(k)}\left(r_{ji}\right) = (-1)^{|\mathsf{L}(j)|} \left(\prod_{i' \in \mathsf{L}(j) \backslash \{i\}} \mathrm{sgn}\left(\mathrm{LLR}^{(k)}\left(q_{ji'}\right)\right) \right) \cdot \min_{i' \in \mathsf{L}(j) \backslash \{i\}} \left(\left| \mathrm{LLR}^{(k)}\left(q_{ji'}\right) \right| \right)$$

- Step 3. **Compute output.** The output message from the decoder is the extrinsic information,

$$\mathrm{LLR}^{\mathrm{ext}}_{\mathrm{LDPC}}\left(v_i\right) = \sum_{j' \in \mathsf{M}(i)} \mathrm{LLR}^{(k)}\left(r_{j'i}\right),$$

  and the estimate of the posterior information $\mathrm{LLR}^{\mathrm{post}}\left(v_i\right)$ is given by

$$\mathrm{LLR}^{(k)}\left(q_i\right) = \sum_{j' \in \mathsf{M}(i)} \mathrm{LLR}^{(k)}\left(r_{j'i}\right) + \mathrm{LLR}\left(p_i\right),$$

  following (2.70). This is used to compute the bit-by-bit estimate of the codeword:

$$\widehat{v}_i^{(k)} = \begin{cases} 1 & \text{if } \mathrm{LLR}^{(k)}\left(q_i\right) > 0 \\ 0 & \text{if } \mathrm{LLR}^{(k)}\left(q_i\right) < 0 \end{cases}$$

- Step 4. **Repeat until done**. Check if the stopping condition has been reached (e.g., a fixed number of iterations $k_{\max}$ has been reached, or the decision word $\widehat{v}^{(k)}$ satisfies the parity-check matrix H). If not, then increment $k \leftarrow k+1$, and repeat Steps 1, 2 (or 2′), and 3.

In the next section, we discuss how to compute the intrinsic LLRs in (2.75). For reference, for common channel models, the intrinsic LLRs can be found as follows:

- for a binary symmetric channel with crossover probability p and received bit y_i,

$$\mathrm{LLR}^{\mathrm{int}}_{\mathrm{LDPC}}\left(v_i\right) = \begin{cases} \log\frac{1-p}{p} & \text{if } y_i = 1 \\ \log\frac{p}{1-p} & \text{if } y_i = 0 \end{cases}$$

- for an AWGN channel with noise variance σ^2 and received signal $\mathbf{y}_i$,

$$\mathrm{LLR}^{\mathrm{int}}_{\mathrm{LDPC}}\left(v_i\right) = \frac{2}{\sigma^2}\mathbf{y}_i.$$

4.4 MEMORYLESS CHANNELS

Now we consider using the LDPC decoder in conjunction with a channel decoder. Consider the situation shown in Figure 2.23 (see also Figure 1.11), where the ECC encoder (e.g., LDPC encoder) first takes a user bit sequence u to an encoded bit sequence v, which is then passed through a mapper that maps the bits to a sequence of symbols x to be transmitted. After these signals pass through the channel, the received sequence y must then be decoded by a channel decoder.

In this section, we consider the three memoryless channels presented in the Introduction (§1-2.1), and consider modules that use the received value y_i to find an extrinsic probability $P^{\mathrm{ext}}_{\mathbf{chan}}\left(x_i\right)$ on the transmitted symbol x_i. (In later chapters (ref. §3-1.3), channels with memory are considered.) This module for handling the received values is a "channel decoder," also known as a "channel detector." For some channels, the operation of this module may turn out to be rather trivial, but we nevertheless continue to call it a "decoder" for consistency.

Following our discussion about modules in Section 2.3, for any edge between modules, the extrinsic probability w.r.t. one module gives the intrinsic probability w.r.t. the other module. In particular, this applies for passing messages between the three modules, the channel decoder, a demapper and an ECC decoder, shown in 2.23. Note that in the binary-input channels considered in this section, the mapper is an assignment

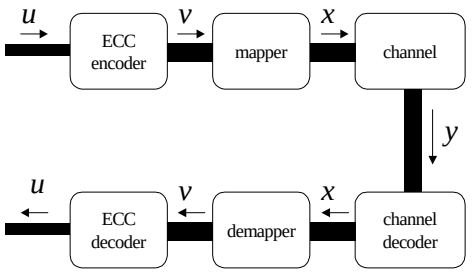

Figure 2.23. Channel decoder, demapper and ECC decoder

of a bit to a binary signal, so that the demapper just needs to pass the messages straight through, acting like an equality module with two bundles.

Starting from the channel decoder, the extrinsic probability $P^{\text{ext}}_{\text{chan}}(x_i)$ is taken as an intrinsic probability by the demapper $P^{\text{int}}_{\text{demap}}(x_i)$, which then (trivially) maps it to a probability $P^{\text{ext}}_{\text{demap}}(v_i)$ for v_i, which is in turn taken as an intrinsic probability by the ECC decoder $P^{\text{int}}_{\text{ECC}}(v_i)$, as in (2.75). (To complete the process, the ECC decoder computes extrinsic probabilities $P^{\text{ext}}_{\text{ECC}}(v_i)$, which can then be used to generate decisions for the user bits u_i.) In this section, we use "ECC" rather than "LDPC" in order to include other error-control codes in this discussion.

4.4.1 BINARY SYMMETRIC CHANNEL

For the binary symmetric channel (BSC), with cross-over probability p, if the input to the channel is a binary variable x_i and the output is a binary variable y_i, then the conditional probabilities introduced by the channel are given by

$$\begin{aligned} P(y_i = a \mid x_i = a) &= 1 - p \\ P(y_i \neq a \mid x_i = a) &= p \end{aligned}$$

for $a = 0, 1$. Then given the received information y_i, the posterior probability can be rewritten using Bayes' theorem,

$$P(x_i = a \mid y_i = b) = \frac{P(y_i = b \mid x_i = a)\, P(x_i = a)}{P(y_i = b)}.$$

Since $P(x_i = a)$ is the intrinsic probability for x_i, the extrinsic probability from this module is proportional to $P(y_i = b \mid x_i = a)$, which in this case gives

$$P^{\text{ext}}_{\text{BSC}}(x_i) = \begin{cases} 1-p & \text{if } y_i = x_i \\ p & \text{if } y_i \neq x_i \end{cases}.$$

In terms of log-likelihood ratios, this gives

$$\text{LLR}^{\text{ext}}_{\text{BSC}}(x_i) = \begin{cases} \log \frac{1-p}{p} & \text{if } y_i = 1 \\ \log \frac{p}{1-p} & \text{if } y_i = 0 \end{cases},$$

which can be used as $\text{LLR}^{\text{int}}_{\text{ECC}}(v_i)$ for input to the soft decoder for the ECC.

Actually, instead of the message-passing algorithm for LDPC codes, there exist hard-decision versions of the decoding algorithm that can take as input the received bits y_i directly from the binary symmetric channel. These hard-decision algorithms, including one introduced by Gallager [Gal63] and another by Sipser and Spielman [SS96], are more appropriate for the binary symmetric channel (and are computationally simpler) than the sum-product update rule.

4.4.2 BINARY ERASURE CHANNEL

On the other hand, for the binary erasure channel (BEC), with erasure probability p, the conditional probabilities introduced by the channel are

$$\begin{aligned} P(y_i = 1 \mid x_i = 1) &= 1 - p \\ P(y_i = E \mid x_i = 1) &= p \\ P(y_i = E \mid x_i = 0) &= p \\ P(y_i = 0 \mid x_i = 0) &= 1 - p \end{aligned}$$

where E is the erasure symbol. Again, the extrinsic probability is proportional to these transition probabilities, so that

$$P^{\text{ext}}_{\text{BEC}}(x_i = 1) = \begin{cases} 1-p & \text{if } y_i = 1 \\ p & \text{if } y_i = E \\ 0 & \text{if } y_i = 0 \end{cases}$$

$$P^{\text{ext}}_{\text{BEC}}(x_i = 0) = \begin{cases} 0 & \text{if } y_i = 1 \\ p & \text{if } y_i = E \\ 1-p & \text{if } y_i = 0 \end{cases}.$$

In terms of log-likelihood ratios, this gives

$$\mathrm{LLR}_{\mathrm{BEC}}^{\mathrm{ext}}(x_i) = \begin{cases} +\infty & \text{if } y_i = 1 \\ 0 & \text{if } y_i = E \\ -\infty & \text{if } y_i = 0 \end{cases}.$$

This reflects the fact that receiving $y_i = 0$ or 1 means that the input had to be $x_i = 0$ or 1, respectively.

Then the inputs to the ECC decoder would be

$$\mathrm{LLR}_{\mathrm{ECC}}^{\mathrm{int}}(v_i) = \mathrm{LLR}_{\mathrm{BEC}}^{\mathrm{ext}}(x_i),$$

which lies in the range $\{-\infty, 0, +\infty\}$. These infinite values may not be suitable for the LDPC decoding algorithm since they cause equations (2.68) and (2.69) to become not well defined. Decoding algorithms designed for the binary erasure channel are presented in Luby *et al.* [LMSS01a].

4.4.3 AWGN CHANNEL

Finally, we consider the AWGN channel, in which additive white Gaussian noise is added to the transmitted signal. The signals are real-valued (which is indicated by the use of variables with bold font), and the received signal $\mathbf{y}_i$ is given by

$$\mathbf{y}_i = \mathbf{x}_i + \mathbf{n}_i,$$

which consists of the transmitted signal $\mathbf{x}_i$ with additive noise $\mathbf{n}_i$ chosen from a zero-mean Gaussian distribution

$$P_n(a) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{a^2}{2\sigma^2}\right),$$

where σ is the standard deviation of the noise. The transmitted signal $\mathbf{x}_i$ is often restricted to a discrete signalling constellation, such as pulse amplitude modulation (PAM)). A complex-valued AWGN channel can also be defined in an analogous way.

We consider the use of a binary PAM constellation, which is called the binary-input AWGN channel (sometimes abbreviated "BIAWGNC"). The encoded bits $v_i \in \{0, 1\}$ are mapped to the constellation points $\mathbf{x}_i \in \{-1, 1\}$ for transmission, using the mapping $\mathbf{x}_i = 2v_i - 1$.[9] Since the capacity of the AWGN channel is achieved by a transmit signal that

[9] It should be noted that the opposite convention, $\mathbf{x}_i = (-1)^{v_i} = 1 - 2v_i$, is commonly used in communications.

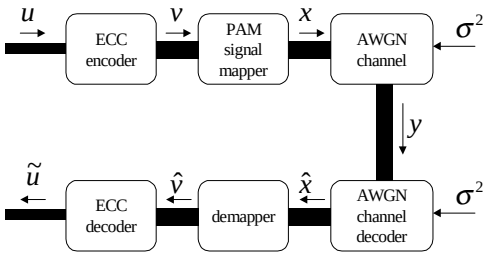

Figure 2.24. Block diagram of system with an AWGN channel

also has a Gaussian distribution, the limitation of the transmitted signal to binary PAM reduces the capacity. The calculation of the capacity of the binary-input AWGN channel is given in Appendix 2.A, and the BI-AWGNC capacity and AWGN channel capacity are compared in Figure 2.25.

In the case of an AWGN channel, where there is no intersymbol interference (ISI), the channel receiver can estimate the channel log-likelihood ratio as follows,

$$\begin{aligned}
\mathrm{LLR}_{\mathrm{AWGN}}^{\mathrm{post}}(\mathbf{x}_i) &= \log\frac{P(\mathbf{x}_i = 1 \mid \mathbf{y}_i)}{P(\mathbf{x}_i = -1 \mid \mathbf{y}_i)} \\
&= \log\frac{P(\mathbf{y}_i \mid \mathbf{x}_i = 1)}{P(\mathbf{y}_i \mid \mathbf{x}_i = -1)} + \log\frac{P(\mathbf{x}_i = 1)}{P(\mathbf{x}_i = -1)} \\
&= \log\frac{\exp\left(-\frac{1}{2\sigma^2}(\mathbf{y}_i - 1)^2\right)}{\exp\left(-\frac{1}{2\sigma^2}(\mathbf{y}_i + 1)^2\right)} + \mathrm{LLR}_{\mathrm{AWGN}}^{\mathrm{int}}(\mathbf{x}_i) \\
&= \frac{2}{\sigma^2}\mathbf{y}_i + \mathrm{LLR}_{\mathrm{AWGN}}^{\mathrm{int}}(\mathbf{x}_i),
\end{aligned}$$

so that the extrinsic log-likelihood ratio with respect to the channel decoder is

$$\mathrm{LLR}_{\mathrm{AWGN}}^{\mathrm{ext}}(\mathbf{x}_i) = \frac{2}{\sigma^2}\mathbf{y}_i. \tag{2.76}$$

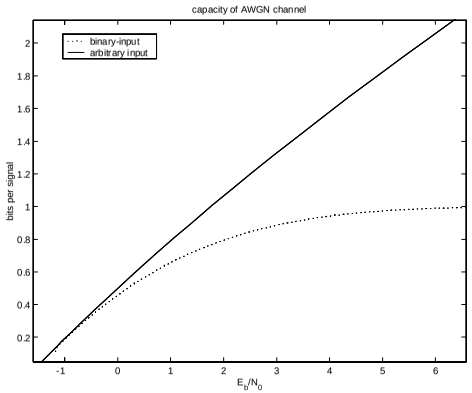

Figure 2.25. Capacities of (arbitrary-input) AWGN channel and binary-input AWGN channel

The demapper simply assigns the probabilities for $\mathbf{x}_i$ to v_i. For a 2-PAM constellation, this would simply consist of assignments of the form

$$\mu_{X_i \to V_i}(v_i = 1) = \mu_{N_i \to X_i}(\mathbf{x}_i = 1)$$
$$\mu_{X_i \to V_i}(v_i = 0) = \mu_{N_i \to X_i}(\mathbf{x}_i = -1).$$

Then the intrinsic LLRs for the ECC decoder are given by

$$\mathrm{LLR}^{\mathrm{int}}_{\mathrm{ECC}}(v_i) = \mathrm{LLR}^{\mathrm{ext}}_{\mathrm{AWGN}}(\mathbf{x}_i) = \frac{2}{\sigma^2}\mathbf{y}_i.$$

In terms of probabilities, this can be written as

$$P^{\mathrm{int}}_{\mathrm{ECC}}(v_i) = \frac{\exp\left(\frac{2}{\sigma^2}\mathbf{y}_i\right)}{1+\exp\left(\frac{2}{\sigma^2}\mathbf{y}_i\right)}. \tag{2.77}$$

These probabilities on v_i can then be taken as the input to the LDPC soft decoding algorithm. (Note that the AWGN channel decoder depends on an estimate for the channel noise parameter σ^2. In practical implementations, this parameter needs to be estimated, and deviations from the actual value of σ^2 may affect the performance. In these simulations, it is assumed that this noise parameter is known perfectly.)

The quality of the channel is measured in terms of the signal to noise ratio (SNR), which is defined as

$$\frac{E_b}{N_0} = \frac{P}{2R\sigma^2}, \tag{2.78}$$

which represents the energy normalized per user bit. The power is assumed to be $P = 1$, the code rate is R , and the noise variance is σ^2. Adjusting by the code rate R in the denominator turns out to give a fair comparison between codes of different rates. The signal-to-noise ratio is usually measured in decibels (dB), so that the SNR is equal to $10\log_{10}\left(\frac{E_b}{N_0}\right)$ dB.

We show several figures giving the performance of a LDPC code on a ISI-free channel with additive white Gaussian noise (AWGN). The default parameters are as follows: the rate of the LDPC code is $R = \frac{8}{9}$, the block length is $N = 2448$, the parity-check matrices are constructed to have $t_c = 3$ bits per column (and no 4-cycles), and the decoding is performed for a maximum of 20 iterations.

Figure 2.26 considers the performance for different block lengths. In addition, the binary-input capacity formula is used to plot a capacity curve showing the minimum SNR that is required to achieve that error rate with a rate $\frac{8}{9}$ code. A code of length $N = 4896$ requires 4.3 dB to achieve a bit error rate of 10^{-5}. While this performance is considerably better than the uncoded system, there is still a gap to the minimum SNR ($E_b/N_0 \approx 3\ dB$) required for reliable transmission over the binary-input AWGN channel with a rate $\frac{8}{9}$ code. The performance can be improved by increasing the block length and optimizing the design of the parity-check matrix.

Figure 2.27 considers LDPC codes of different rates R. Note that the definition of E_b/N_0 has already taken into account the differences in rate, so that the figure gives a fair comparison of the different coding rates.

Figure 2.28 considers the effect of changing the design of the parity-check matrix to have $t_c = 2, 3, 4$ bits per column. It can be seen that having only $t_c = 2$ bits per column does not work very well, while $t_c = 4$ performs slightly better than $t_c = 3$.

Figure 2.29 plots the bit error rate curve after different numbers of iterations. It can be seen that in this case, even one iteration of the LDPC decoding algorithm gives significant coding gain over an uncoded system, and that allowing a maximum of 4 iterations gets within 0.5 dB of the performance of 20 iterations.

Figure 2.30 shows the block error rate as a function of the SNR, for different numbers of iterations, as seen in . The block error rate curves are steeper than the bit error rate curves.

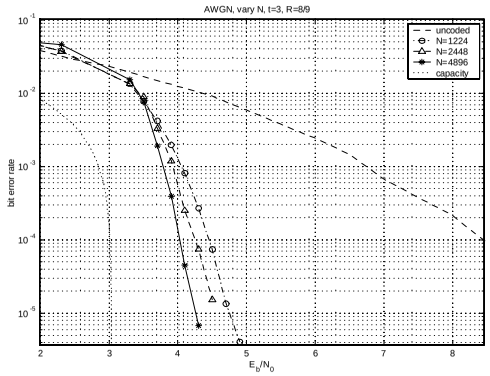

Figure 2.26. Varying the block length N

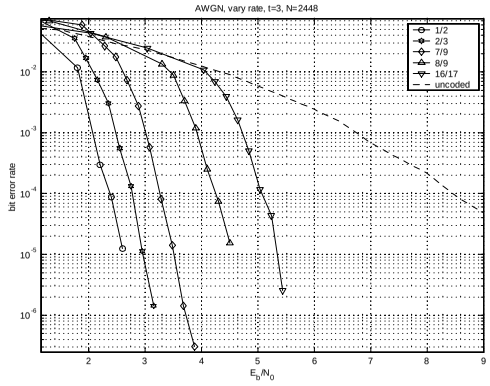

Figure 2.27. Varying the rate R

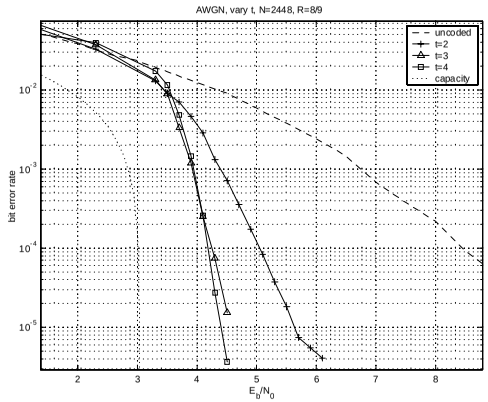

Figure 2.28. Varying the number of bits per column t_c

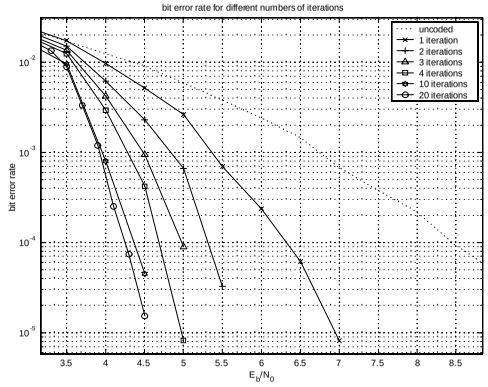

Figure 2.29. Bit error rate vs. SNR, plotted for different numbers of iterations

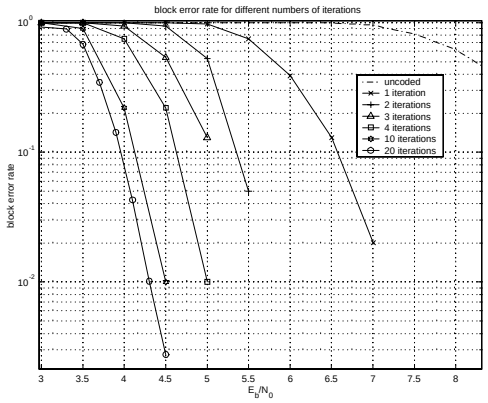

Figure 2.30. Block error rate vs. SNR, plotted for different numbers of iterations

4.5 NUMERICAL EXAMPLE

To illustrate the decoding of LDPC codes, let us consider the following parity check matrix:

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Suppose that the input message is $u = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$. Then taking the first three bits as the systematic message bits, the corresponding encoded codeword is

$$v = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

The transmitted word is then

$$\mathbf{x} = \begin{bmatrix} 1 & 1 & -1 & -1 & 1 & 1 \end{bmatrix}.$$

Suppose that due to noise on an AWGN channel, the received vector is

$$\mathbf{y} = \begin{bmatrix} -1 & 2 & -2 & -2 & 2 & 0 \end{bmatrix}.$$

The extrinsic probability w.r.t. the AWGN decoder is given by (2.76), which goes through the demapper to give the intrinsic probabilities w.r.t. the LDPC decoder:

$$\mathrm{LLR}^{\mathrm{int}}_{\mathrm{LDPC}}(v_i) = \mathrm{LLR}^{\mathrm{ext}}_{\mathrm{chan}}(\mathbf{x}_i) = \frac{2}{\sigma^2}\mathbf{y}_i$$

Then assuming that the AWGN channel is known to have a noise parameter of $\sigma^2 = 2$, for this received word $\mathbf{y}$, the input to the LDPC decoder has a log-likelihood ratio of

$$\mathrm{LLR}(p_i) = \mathrm{LLR}^{\mathrm{int}}_{\mathrm{LDPC}}(v_i) = \begin{bmatrix} -1 & 2 & -2 & -2 & 2 & 0 \end{bmatrix},$$

corresponding to the probabilities

$$p_i = \begin{bmatrix} 0.268 & 0.881 & 0.119 & 0.119 & 0.881 & 0.5 \end{bmatrix}.$$

The first bit has been corrupted, and the last bit has been erased.

4.5.1 MESSAGE-PASSING ALGORITHM

Using the matrix H and the intrinsic information $\mathrm{LLR}(p_i)$, we can apply the log-domain version of the message-passing algorithm. Since $\mathrm{LLR}^{(0)}(r_{ji})$ is initially set to zero, applying equation (2.68) gives the bit-to-check messages $\mathrm{LLR}^{(1)}(q_{ji})$ as $\mathrm{LLR}(p_i)$.

$$\mathrm{LLR}(q_{ji}) = \begin{bmatrix} -1 & 2 & & -2 & & \\ & 2 & -2 & & 2 & \\ -1 & & -2 & & & 0 \end{bmatrix}$$

Notice that the array for $\mathrm{LLR}(q_{ji})$ is left blank when the corresponding entry of matrix H is zero. The dependencies of the entries are shown in Figure 2.31.

Then the first check-to-bit messages $\mathrm{LLR}^{(1)}(r_{ji})$ are computed using equation (2.69), where each message $\mathrm{LLR}(r_{ji})$ is a function of all the other messages $\mathrm{LLR}(q_{ji'})$ in the same row. For example, the first entry in the first row is given by

$$\begin{aligned} \mathrm{LLR}(r_{11}) &= (-1)\,2\tanh^{-1}\left(\tanh\left(\frac{1}{2}\cdot 2\right)\tanh\left(\frac{1}{2}\cdot -2\right)\right) \\ &= 2\tanh^{-1}(0.762^2) = 1.325. \end{aligned}$$

Similarly, it is possible to compute the rest of the messages from checks to bits.

$$\mathrm{LLR}^{(1)}(r_{ji}) = \begin{bmatrix} 1.325 & -0.735 & & 0.735 & & \\ & 1.325 & -1.325 & & 1.325 & \\ 0 & & 0 & & & -0.735 \end{bmatrix}$$

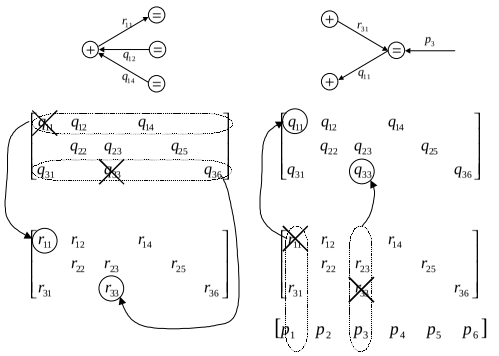

Figure 2.31. Iterating probabilities in the LDPC decoding algorithm (start at the upper left matrix and proceed counter-clockwise)

Then summing up the columns gives the extrinsic output for the first iteration of the decoder, and adding the intrinsic inputs $\mathrm{LLR}(p_i)$ as in equation (2.70) gives the posterior information as

$$\mathrm{LLR}^{(1)}(q_i) = \begin{bmatrix} 0.325 & 2.590 & -3.325 & -1.265 & 3.325 & -0.735 \end{bmatrix}.$$

Making a hard-decision based on the signs of these LLRs as follows,

$$\widehat{v}_i = \begin{cases} 1 & \text{if } \mathrm{LLR}(q_i) > 0 \\ 0 & \text{if } \mathrm{LLR}(q_i) < 0 \end{cases},$$

gives an initial estimate of the transmitted codeword, after one iteration of message-passing:

$$\widehat{v}^{(1)} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Comparing with the original codeword $v = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$, we see that the first bit has been corrected, but the last bit is still in error.

Proceeding with another iteration of the message-passing algorithm, equation (2.68) computes the message $\mathrm{LLR}^{(2)}(q_{ji})$ by summing all the messages $\mathrm{LLR}^{(1)}(r_{j'i})$ in the same column (except for the one in the j-th row) and also adding the intrinsic information $\mathrm{LLR}(p_i)$. As an example, the first entry $\mathrm{LLR}^{(2)}(q_{11})$ can be found by summing over all

$\mathrm{LLR}^{(1)}(r_{j'1})$ in the first column, except for $\mathrm{LLR}^{(1)}(r_{11})$ (since the incoming intrinsic message must be excluded when computing the extrinsic message).

$$\begin{aligned} \mathrm{LLR}^{(2)}(q_{11}) &= \sum_{j' \neq 1} \mathrm{LLR}^{(1)}(r_{j'1}) + \mathrm{LLR}(p_1) \\ &= \mathrm{LLR}^{(1)}(r_{31}) + \mathrm{LLR}(p_1) \\ &= 0 + (-1) = -1 \end{aligned}$$

Continuing this computation, all the messages from bits to checks can be updated as follows:

$$\mathrm{LLR}^{(2)}(q_{ji}) = \begin{bmatrix} -1 & 3.325 & & -2 & \\ & 1.264 & -2 & & 2 \\ 0.325 & & -3.325 & & 0 \end{bmatrix}$$

Note that if the posterior information $\mathrm{LLR}^{(1)}(q_i)$ has already been calculated, then the bit-to-check messages can also be found as

$$\mathrm{LLR}^{(2)}(q_{ji}) = \mathrm{LLR}^{(1)}(q_i) - \mathrm{LLR}^{(1)}(r_{ji}).$$

Next, applying equation (2.69) gives an updated set of check-to-bit messages:

$$\mathrm{LLR}^{(2)}(r_{ji}) = \begin{bmatrix} 1.769 & -0.735 & & 0.920 & & \\ & 1.325 & -0.911 & & 0.911 & \\ 0 & & 0 & & & 0.302 \end{bmatrix}$$

Applying equation (2.70) for the second time yields

$$\mathrm{LLR}^{(2)}(q_i) = \begin{bmatrix} 0.769 & 2.590 & -2.911 & -1.080 & 2.911 & 0.302 \end{bmatrix}.$$

Taking hard decisions then gives a correct estimate of the transmitted codeword,

$$\widehat{v}^{(2)} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

This short parity-check code has corrected an error and an erasure using two iterations of the message-passing algorithm.

4.5.2 "MIN" APPROXIMATION

On the other hand, the "min" approximation in (2.74) offers similar decoding performance with simplified implementation. We start again with the intrinsic probabilities from the channel, which yields the initial

messages q_{ji} from bits to checks.

$$\mathrm{LLR}\left(p_i\right) = \begin{bmatrix} -1 & 2 & -2 & -2 & 2 & 0 \end{bmatrix}$$

$$\mathrm{LLR}^{(1)}\left(q_{ji}\right) = \begin{bmatrix} -1 & 2 & & -2 & & \\ & 2 & -2 & & 2 & \\ -1 & & -2 & & & 0 \end{bmatrix}$$

Applying (2.74) yields the messages from checks to bits using the minimum function:

$$\mathrm{LLR}^{(1)}\left(r_{ji}\right) = \begin{bmatrix} 2 & -1 & & 1 & & \\ & 2 & -2 & & 2 & \\ 0 & & 0 & & & -1 \end{bmatrix}$$

Summing up the columns of $\mathrm{LLR}^{(1)}\left(r_{ji}\right)$ as well as the intrinsic information $\mathrm{LLR}\left(p_i\right)$ using (2.70) gives the posterior LLR as

$$\mathrm{LLR}^{(1)}\left(q_i\right) = \begin{bmatrix} 1 & 3 & -4 & -1 & 4 & -1 \end{bmatrix}.$$

Taking hard-decisions gives the vector $\widehat{v}^{(1)} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$, which is incorrect in the last bit.

The next iteration begins by applying (2.68) to obtain the updated messages from bits to checks.

$$\mathrm{LLR}^{(2)}\left(q_{ji}\right) = \begin{bmatrix} -1 & 4 & & -2 & & \\ & 1 & -2 & & 2 & \\ 1 & & -4 & & & 0 \end{bmatrix}$$

Then applying (2.74) once more gives

$$\mathrm{LLR}^{(2)}\left(r_{ji}\right) = \begin{bmatrix} 2 & -1 & & 1 & & \\ & 2 & -1 & & 1 & \\ 0 & & 0 & & & 1 \end{bmatrix},$$

so that summing the columns of $\mathrm{LLR}^{(2)}\left(r_{ji}\right)$ and $\mathrm{LLR}\left(p_i\right)$ gives

$$\mathrm{LLR}^{(2)}\left(q_i\right) = \begin{bmatrix} 1 & 3 & -3 & -1 & 3 & 1 \end{bmatrix},$$

which corresponds to the correct codeword after making a hard-decision:

$$\widehat{v}^{(2)} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

APPENDIX 2.A: Capacity of binary-input channel

The evaluation of the capacity of a binary-input channel with additive white Gaussian noise is as follows. (This description is taken from

[Fre98]). First, for a given noise level, represented by the noise variance σ^2, the mutual information is given by

$$\begin{aligned} M\left(\sigma^2\right) = I\left(x_i; y_i\right) &= \sum_{x_i \in A} \int_{y_i} P\left(x_i, y_i\right) \log_2 \frac{P\left(x_i, y_i\right)}{P\left(x_i\right) P\left(y_i\right)} dy_i \\ &= \sum_{x_i \in A} \int_{y_i} P\left(x_i, y_i\right) \log_2 P\left(y_i \mid x_i\right) dy_i - \int_{y_i} P\left(y_i\right) \log_2 P\left(y_i\right) dy_i \end{aligned}$$

where $A = \{-1, +1\}$ is the PAM constellation for binary signalling. The first term is the negative of the entropy of a Gaussian distribution, $-\frac{1}{2}\log_2\left(2\pi\sigma^2 e\right)$, while the second term is given by

$$\int_{y_i} \left(\sum_{x_i \in A} \frac{\exp\left(-\frac{1}{2\sigma^2}\left(y_i - x_i\right)^2\right)}{2\sqrt{2\pi\sigma^2}} \right) \log_2 \left(\sum_{x_i \in A} \frac{\exp\left(-\frac{1}{2\sigma^2}\left(y_i - x_i\right)^2\right)}{2\sqrt{2\pi\sigma^2}} \right) dy_i$$

which can be evaluated using numerical integration.

Then the mutual information $M\left(\sigma^2\right)$ gives the capacity, which represents the maximum rate R that allows reliable transmission. Using the definition (2.78), the minimum signal-to-noise ratio (E_b/N_0) that allows reliable transmission is then given by

$$\mathrm{SNR}_{\min} = \frac{1}{2M\left(\sigma^2\right)\sigma^2},$$

where $P = 1$, assuming binary PAM.

Chapter 8

OTHER TOPICS

We close with some topics related to the application of soft iterative decoding techniques to communications.

1. MULTILEVEL MODULATION

To extend these ideas of soft iterative decoding to other communications channels, it is necessary to consider multilevel modulation. In the context of communications, modulation refers to how the signal is shaped for transmission over a channel. The modulation consists of assigning the bits to some constellation points, which represent the signalling waveforms that are used to transmit the signal. We restrict our attention to Pulse Amplitude Modulation (PAM), in which the signal constellations are points on the real line, since Quadrature Amplitude Modulation (QAM) in the complex plane can often be implemented so that the real and imaginary components may be decoded separately.

To encode these constellations, there are various possibilities for assigning bits to the constellation points. If we then apply an error-control code on these bits, this configuration is known as coded modulation. If we use Gray coding for the labels, so that labels for adjacent constellation points only differ in a single bit, then this type of coded modulation is called *Bit-Interleaved Coded Modulation* (BICM). (If we use another labelling that divides the constellation into cosets, and then apply a convonultional code on the bits that choose the coset, this is called *Trellis-Coded Modulation* (TCM).) The details of the situation will determine the type of coded modulation that is appropriate. (Refer to [Che97][NLi00] for more details on BICM and TCM with soft iterative decoding.)

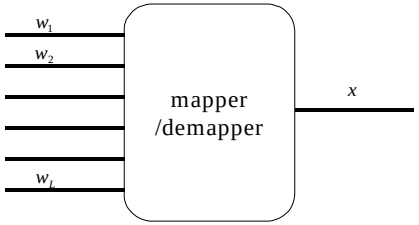

Figure 8.1. Module for mapping and demapping

The demapper then has the task of finding the extrinsic probabilities based on the received signals. Until now, we have only considered binary PAM (where $w = \{0, 1\}$ maps to $\mathbf{x} = \{-1, 1\}$), so that the mapping and demapping have been trivial.

For the general case, as shown in Figure 8.1, we can treat this signal mapper as a local constraint S, which puts a restriction on the input bits $(w_1, w_2, ..., w_L)$ and the output signal $\mathbf{x}$, which is a real-valued constellation point (in the case of PAM) or a complex constellation point (in the case of QAM). In other words, for each set of input bits, there is a unique output $\mathbf{x}$ corresponding to the point that is labelled by these input bits.

This gives us the description of the demapper, whose update equation is as follows,

$$P^{\text{ext}}_{\text{demap}}(w_i) = c'_{w_i} \sum_{\substack{(w_1,...,w_L,\mathbf{x})\in S \\ \sim\{w_i\}}} P^{\text{int}}_{\text{demap}}(\mathbf{x}) \prod_{i'\neq i} P^{\text{int}}_{\text{demap}}(w_{i'}), \qquad (8.1)$$

where c'_{w_i} is a normalizing constant. If there are no intrinsic probabilities on the bits (i.e. the intrinsic probabilities are all equal), then the expression becomes simpler yet:

$$P^{\text{ext}}_{\text{demap}}(w_i) = c''_{w_i} \sum_{\substack{(w_1,...,w_L,\mathbf{x})\in S \\ \sim\{w_i\}}} P^{\text{int}}_{\text{demap}}(\mathbf{x}) \qquad (8.2)$$

In the case of 4-PAM, the constraint set S might consist of elements $(w_1, w_2, \mathbf{x})$ as follows:

$$S = \{(0, 0, +3), (1, 0, +1), (1, 1, -1), (0, 1, -3)\}$$

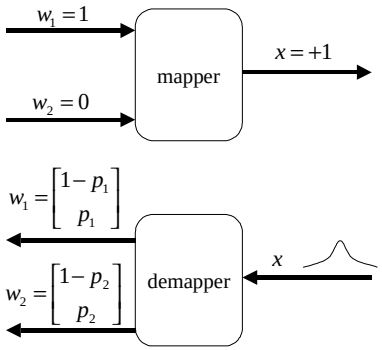

Figure 8.2. Mapping and demapping for 4-PAM

In this case, the label uses Gray coding, so that this demapper corresponds to bit-interleaved coded modulation (BICM). Mapping and soft demapping in this example is depicted in Figure 8.2.

In the case of an AWGN channel, then the probability density function on $\mathbf{x}$ will be a Gaussian distribution. In other words, if the value $\mathbf{y}$ was received, and it is known that the noise has variance σ_y^2, then $\mathbf{x}$ has a Gaussian distribution with mean value equal to $\mathbf{y}$ and variance σ_y^2.

$$P_{\text{demap}}^{\text{int}}(\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma_y} \exp\left(-\frac{1}{2\sigma_y^2}(\mathbf{x}-\mathbf{y})^2\right).$$

In the expression for the extrinsic probability (8.2), the largest term will tend to dominate the summation. In other words, the PAM constellation point that is closest to the received signal $\mathbf{y}$ will be the dominating term,

$$\begin{aligned} P^{\text{ext}}(w_i = \omega_i) &= c_{w_i} \sum_{\substack{(w_1,\ldots,w_L,\mathbf{x})\in S \\ \sim\{w_i\}}} P^{\text{int}}(\mathbf{x}) \\ &\approx c_{w_i} \frac{1}{\sqrt{2\pi}\sigma_y} \exp\left(-\frac{1}{2\sigma_y^2}(X_{y,w_i=\omega_i} - \mathbf{y})^2\right) \end{aligned}$$

where the point

$$X_{y,w_i=\omega_i} = \arg \min_{\substack{\mathbf{x} \text{ such that} \\ (w_1,\ldots,w_L,\mathbf{x})\in S \\ \text{and } w_i=\omega_i}} (\mathbf{x}-\mathbf{y})^2$$

represents the closest point $\mathbf{x}$ to the received signal $\mathbf{y}$ out of all the points in the PAM constellation that have labels with bit w_i equal to ω_i. Then the log-likelihood ratio is given as follows,

$$\text{LLR}^{\text{ext}}(w_i) = \frac{\exp\left(-\frac{1}{2\sigma_y^2}(X_{y,w_i=1} - \mathbf{y})^2\right)}{\exp\left(-\frac{1}{2\sigma_y^2}(X_{y,w_i=0} - \mathbf{y})^2\right)},$$

which can be simplified as

$$\begin{aligned} \text{LLR}^{\text{ext}}(w_i) = -\frac{1}{2\sigma_y^2}\Big((X_{y,w_i=1})^2 - (X_{y,w_i=0})^2 \\ + 2(-X_{y,w_i=0} + X_{y,w_i=1})\,\mathbf{y}\Big). \end{aligned}$$

(Note that for 2-PAM, this expression simplifies to

$$\text{LLR}^{\text{ext}}(w_1) = \frac{2}{\sigma_y^2}\mathbf{y}$$

since $X_{y,w_i} = \pm 1$, which coincides with the expression derived in (2.76).)

As a result of this approximation, the log-likelihood ratio is simply a linear function of the received signal $\mathbf{y}$, making the complexity of this approximate soft demapping operation feasible for practical implementations. In addition, we mention that it is possible to iterate soft information with this demapper. In other words, updated extrinsic probabilities for w from the ECC decoding could be used as intrinsic probabilities for the demapper to improve the demapping, using equation (8.1). In this way, the demapper can also be integrated into the soft iterative decoding framework for additional coding gain, as demonstrated in Muquet *et al.* [MMDC00].

2. POTENTIAL APPLICATIONS OF CONSTRAINED CODING

We briefly list some situations other than the storage channel where the ideas related to constrained coding may potentially be applicable:

- Constellation-shaping codes that choose the distribution of constellation points such that the resulting distribution of the transmit signal