Graphical Models and Iterative Decoding

Thesis by

Srinivas M. Aji

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

2000
(Submitted May 12, 2000)

Acknowledgements

I thank my advisor Prof. McEliece for his guidance and support throughout. I thank Padhraic Smyth for introducing us to the newer developments in Bayesian networks.

I also thank the other occupants of Moore 155 for making it a more pleasant place by being there.

Abstract

Since the invention of turbo codes, there has been a lot of interest in iterative decoding schemes. It is also known that the turbo decoding algorithm and several other previously known iterative algorithms are instances of Pearl's belief propagation algorithm applied to a graph with cycles, while the algorithm is known to work only for graphs without cycles. We describe a marginalization algorithm which works on junction trees, which is based on some newer developments in Bayesian networks. This is sufficiently general that Pearl's belief propagation and decoding on Tanner graphs may be regarded as special cases. An attempt to compute the discrete Fourier transform as a marginalization problem in this framework gives the fast Fourier transform algorithm, thus showing that this framework has applications apart from probabilistic computations. Junction graphs with cycles lead to an iterative algorithm. The case of junction graphs with a single cycle is analyzed, with specific results in the case of the sum-product algorithm. We also have some experimental results for small turbo code-like junction graphs.

On a different topic, we consider the typical set decoder, which can be used to obtain bounds on the noise threshold for asymptotically error free decoding, for given code ensembles. Some choices of the typical set for AWGN channel are considered and the resulting bounds on the threshold obtained.

Contents

List of Figures

Chapter 1 Introduction

There are now several examples of decoding algorithms which can be expressed as message passing algorithms on a graph [12, 29, 28, 33, 23]. These algorithms perform maximum likelihood decoding when the graph is acyclic. When the graph has cycles, we have an iterative algorithm which is found experimentally to do well in terms of decoding performance in several cases [12, 7, 19, 20], and are computationally feasible in contrast with exact maximum likelihood algorithms for the corresponding codes. We present a unified framework for these algorithms in terms of a message passing algorithm on junction graphs [15, 27] which we call the "generalized distributive law."

1.1 Decoding and Probabilistic Inference

The canonical communication systems picture is given in Figure 1.1. The string of k symbols U is encoded into a string of n symbols X which is then transmitted over the channel. At the output of the channel, the string Y is received, from which the decoder has to compute $\tilde{U}$, an estimate of the transmitted information U. If the decoder has to minimize the average probability of symbol error, then the ith symbol of $\tilde{U}$ is chosen as the u_i which maximizes the probability $p(u_i|Y)$. If the decoder has to minimize the probability of word error, then $\tilde{U}$ is chosen to be the string U which maximizes $p(U|Y)$.

The probabilistic model for the communication system is as follows. The information symbols U are generated according to some probability distribution $p(U)$ which is usually uniform. We assume that each symbol is generated independently so that $p(U) = \prod_i p(u_i)$. The encoder encodes U deterministically to X so that $p(X|U)$ is equal to 1 if X is the encoded version of U and 0 otherwise. The channel is described by its conditional probabilities $p(Y|X)$. If the channel is memoryless, the output at each time depends only on the input at each time so that $p(Y|X) = \prod_j p(y_j|x_j)$.

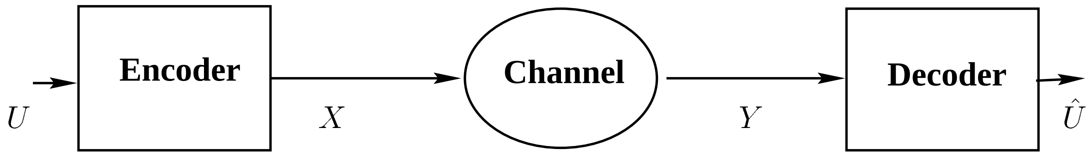

Figure 1.1: Parts of a communication system.

Thus we can write the joint probability

$$\begin{aligned} p(U, X, Y) &= p(U)p(X|U)p(Y|X) & (1.1)\\ &= \prod_i p(u_i)p(X|U)\prod_j p(y_j|x_j) & (1.2) \end{aligned}$$

for independent information symbols and a memoryless channel.

The conditional probabilities $p(u_i|Y)$ and $p(U|Y)$ are proportional to $p(u_i, Y)$ and $p(U, Y)$, with Y fixed at its received value. The latter functions can be computed as marginals of the joint probability $p(U, X, Y)$. Thus the decoding problem is reduced to finding the marginalization of a function of several variables, which often factors into several parts.

We need not consider the input bits U but merely work with the codeword X as follows. Let $\phi(X)$ be a function proportional to the probability that codeword X is transmitted, and let it be zero if X is not a codeword. If the codewords are all equally likely, then $\phi(X)$ can be taken to be the indicator function of the set of codewords. Then we want to find the values for all the x_i which maximize each $p(x_i|Y)$ to minimize the symbol error probability or maximize $p(X|Y)$ to minimize the block error probability. The expression

$$\hat{x}_i = \arg\max_{x_i} \sum_{\{x_j\}_{j\neq i}} p(Y|X)\phi(X) \qquad (1.3)$$

gives the value for x_i maximizing $p(x_i|Y)$, and

$$\hat{x}_i = \arg\max_{x_i} \max_{\{x_j\}_{j \neq i}} p(Y|X)\phi(X). \tag{1.4}$$

gives the value for x_i which maximizes $p(X|Y)$.

We have seen that $p(Y|X)$ factorizes as $\prod_j p(y_j|x_j)$ for a memoryless channel. If the code is constructed so that X is a codeword iff it satisfies each of a set of conditions, then the indicator function for the set of codewords can be written as the product of the indicator function for each of these conditions, and then

$$\phi(X) = \prod_l \phi_l(X), \tag{1.5}$$

where $\phi_l(X)$ is 1 if the lth condition is satisfied by the string X and 0 otherwise. In cases where each of these conditions involves only a small number of the x_i, this factorization may lead to an efficient algorithm for finding the x_i which maximizes $p(x_i|Y)$ or $p(X|Y)$, as described in chapter 2.

1.2 Outline

In chapter 2 we describe a message passing graphical algorithm to solve marginalization problems. We were motivated by the similarity between Pearl's belief propagation algorithm [26] and Wiberg's [33] generalization of the Gallager-Tanner algorithm to express these as special cases of a general algorithm. Using the post-Pearl developments [15, 27], we have a graphical algorithm based on message passing which not only generalizes these two algorithms but provides a unified framework for several marginalization problems.

The message passing algorithm gets its efficiency by reordering computations using the distributive law. Hence we call it the "Generalized Distributive Law." Motivated by the observation that the Fast Fourier Transform is also based on reorganizing computations using the distributive law, we found that the Fast Fourier Transform

[25] can be expressed in the framework of the generalized distributive law. This is described in chapter 3.

We then discuss the notion of junction graphs, which generalizes the notion of junction trees to allow cycles in the graph. As the simplest example of the message passing algorithm on junction graphs with cycles, the case of single cycle junction graphs turns out to be possible to analyse and can be expressed in terms of matrix multiplications. This is described in chapter 4. This lets us apply some results about matrices to obtain information about the behaviour of the iterative algorithm on a single cycle graph.

Chapter 5 describes some numerical simulations of the iterative algorithm on some small graphs based on a simplified junction tree representation of “turbo” like codes. The results support the conjecture that the result of the iterative algorithm is, *in some way*, close to that of the exact algorithm.

On a different subject from the rest, chapter 6 describes typical set decoding on the AWGN channel and considers several choices for the decoder, giving us corresponding bounds on the noise thresholds for asymptotically error free decoding.

Chapter 2 Junction Trees and the Generalized Distributive Law

While attempting to generalize the belief propagation algorithm for Bayesian networks [26] and the graphical decoding algorithm in [33], we found a fairly general framework for doing marginalizations of factored functions in [15, 27]. In this chapter, we present this framework from a more abstract viewpoint than probability propagation. The new results presented are a more careful consideration of the complexity of the algorithm and the use of shortest spanning tree algorithm to construct a junction tree with a set of kernels, where one exists. The message passing algorithm presented here derives its efficiency from reorganizing computations using the distributive law. For example using the identities $a \cdot b_1 + a \cdot b_2 = a \cdot (b_1 + b_2)$ or $a_1 \cdot b_1 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_2 \cdot b_2 = (a_1 + a_2) \cdot (b_1 + b_2)$ can reduce the number of additions and multiplications. For this reason, we call this framework the "Generalized Distributive Law." Most of this chapter has been written up in [3].

2.1 Definitions and Notation

Definition. We define a *commutative semiring* to be a triple $(R, +, \cdot)$ where R is a set and $+ : R \times R \to R$ and $\cdot : R \times R \to R$ are commutative associative binary operations on R satisfying the additional property that $\cdot$ distributes over $+$, i.e., $a \cdot (b + c) = a \cdot b + a \cdot c$ for all $a, b, c \in R$. For our purposes, it is also convenient if the semiring has multiplicative and additive identities.

Examples.

•Any commutative ring. We shall often be dealing with $(\mathbb{R}, +, \cdot)$, where $\mathbb{R}$ is the set of real numbers.

•$(\mathbb{R}^+, \max, \cdot)$, where $\mathbb{R}^+$ is the set of nonnegative real numbers. This is the semiring we work in for finding maximum probability assignments to a set of variables. An isomorphic semiring is $(\mathbb{R} \cup \{\infty\}, \min, +)$ with $x \mapsto -\log(x)$ giving the isomorphism between the two.

•Any boolean algebra. In particular, for any set S, $(2^S, \cup, \cap)$ is a commutative semiring. (2^S is the set of all subsets of S.)

For decoding applications we shall be interested in the first two of the above examples. The actions of a decoder given by equations 1.3 and 1.4 correspond to marginalizing a product of functions in the semirings $(\mathbb{R}, +, \cdot)$ and $(\mathbb{R}^+, \max, \cdot)$ respectively. We will usually use the isomorphic $(\mathbb{R} \cup \{\infty\}, \min, +)$ instead of $(\mathbb{R}^+, \max, \cdot)$ since it is more familiar from the operation of the Viterbi algorithm. We now set up some notation for the marginalization problem.

Let $x_1, x_2, \ldots, x_n$ be variables taking values in the finite alphabets $A_1, A_2, \ldots, A_n$. For any $S \subset \{1, \ldots, n\}$, let $A_S = \prod_{i \in S} A_i$ (cartesian product) and x_S be the list of variables $(x_i)_{i \in S}$. Let $A = A_{\{1,\ldots,n\}}$ and $x = x_{\{1,\ldots,n\}} = (x_1, \ldots, x_n)$. We are given $S_1, S_2, \ldots, S_M$ which are subsets of $\{1, \ldots, n\}$ and for each $i = 1, \ldots, M$ we are given functions $\alpha_i : A_{S_i} \to R$ where R is a commutative semiring. We call the sets S_i *local domains*, and we call α_i the *local functions* or *local kernels*. Let $\beta : A \to R$ be defined as

$$\beta(x) = \prod_{i=1}^{M} \alpha_i(x_{S_i}).$$

We call this the *global function* or *global kernel*. We are required to compute the marginalizations of the global function, i.e., we have to compute $\beta_i : A_{S_i} \to R$ where

$$\beta_i(x_{S_i}) = \sum_{x_{S_i^c} \in A_{S_i^c}} \beta(x).$$

Examples.

•One example of such a marginalization problem is the decoding problem in the form of equations 1.3 and 1.4. A specific example is finding the maximum likelihood codeword for the Hamming code. This is a $(7, 4, 3)$ binary linear code with parity check matrix

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

Suppose the vector $(y_1, y_2, \ldots, y_7)$ was received, and we would like to find the most probable codeword $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_7)$.

We use equation 1.4 modified for the $(\mathbb{R} \cup \{\infty\}, \min, +)$ semiring. i.e,

$$\hat{x}_i = \arg\min_{x_i} \min_{\{x_j\}_{j \neq i}} \sum_j [(-\log p(y_j|x_j)) + \phi'(X)], \text{ for } i = 1, \ldots, 7, \tag{2.1}$$

where $\phi'(X)$ is zero if X is a codeword and ∞ otherwise.

In our example, since X is a codeword only if it satisfies the three parity check equations, we can write

$$\phi'(X) = \chi(x_4, x_5, x_6, x_7) + \chi(x_2, x_3, x_6, x_7) + \chi(x_1, x_3, x_5, x_7), \tag{2.2}$$

where χ is a function which is zero if its arguments sum to zero (mod 2) and ∞ otherwise.

Thus equation 2.1 is a marginalization problem in the $(\mathbb{R} \cup \{\infty\}, \min, +)$ semiring with the following local domains and local kernels.

local domain	local kernel
$\{x_1\}$	$-\log p(y_1\|x_1)$
$\vdots$	$\vdots$
$\{x_7\}$	$-\log p(y_7\|x_7)$
$\{x_1, x_3, x_5, x_7\}$	$\chi(x_1, x_3, x_5, x_7)$
$\{x_2, x_3, x_6, x_7\}$	$\chi(x_2, x_3, x_6, x_7)$
$\{x_4, x_5, x_6, x_7\}$	$\chi(x_4, x_5, x_6, x_7)$.

•Another example we consider is that of a probabilistic state machine. At each time $t \in \mathbb{Z}$, this has a state s_t, an input u_t and an output y_t. The u_t are probabilistically

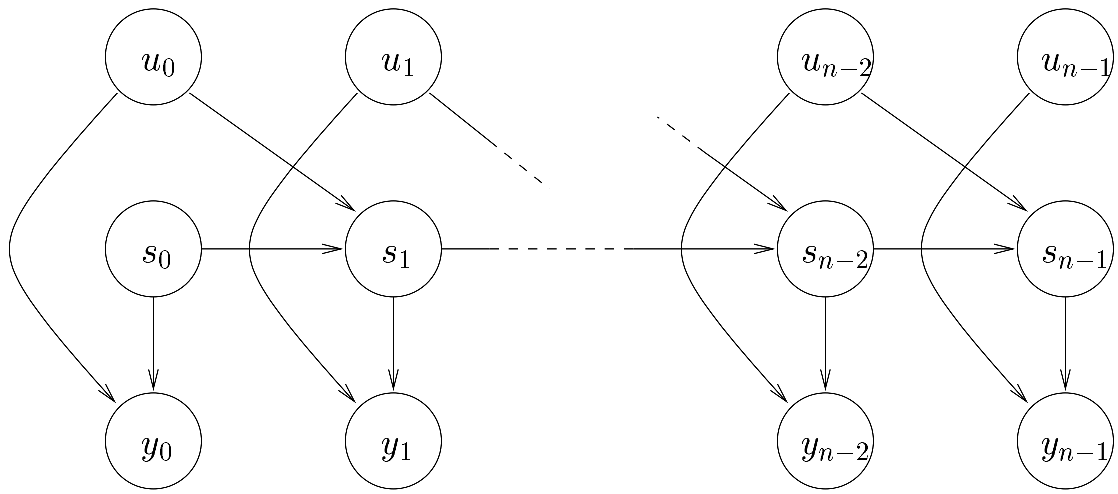

Figure 2.1: Bayesian network for a probabilistic state machine.

generated, independently, with probabilities $p(u_t)$. The output y_t depends on the state s_t and input u_t at that time and is described by the conditional probability distribution $p(y_t|s_t, u_t)$. The state transition process is described by $p(s_{t+1}|s_t, u_t)$, the conditional probability of the next state given the present state and input. If the *a priori* distribution of the initial state s_0 is known, we can write the joint probability of the inputs, state and outputs from time 0 to $n-1$ as

$$p(u_0, \ldots, u_{n-1}, s_0, \ldots, s_{n-1}, y_0, \ldots, y_{n-1}) =$$

$$p(s_0)p(u_0)p(y_0|s_0, u_0) \prod_{t=1}^{n-1} p(s_t|s_{t-1}, u_{t-1})p(u_t)p(y_t|s_t, u_t). \tag{2.3}$$

This factorization by conditional probabilities is described graphically by the Bayesian network [15] in Figure 2.1.

If we observe values of y_t from such a system and want to find the probability of each u_t given our observations, we can use the the joint probability in equation 2.3 with the observed values of y_t and marginalize out all the s_t and also all but one of the u_t. This is thus an instance of the marginalization problem described earlier.

This model also describes convolutional codes, with the following additional remarks. The state transition is deterministic, which means that $p(s_{t+1}|s_t, u_t)$ is one only when $s_{t+1} = s_{t+1}(s_t, u_t)$ and zero otherwise. The output y_t is probabilistically dependent, due to the channel, upon x_t which is a deterministic function of the state and input, and so $p(y_t|s_t, u_t) = p(y_t|x_t(s_t, u_t))$. Marginalizing the product of functions in equation 2.3 in the $(\mathbb{R}, +, \cdot)$ and $(\mathbb{R}^+, \max, \cdot)$ semirings will then give us the maximum likelihood input symbols or input block respectively. The algorithm described in this chapter for such marginalization then gives us essentially the BCJR [6] and Viterbi [29, 9] algorithms respectively.

2.2 Junction Trees and a Message Passing Algorithm

In many cases it is possible to find an algorithm for computing the required marginalizations considerably more efficiently than by direct computation of $\beta(x)$ and summing out the variables not required. This is done by reorganizing the sum of product computation using the distributive law so that various sums are performed as early as possible, before further multiplications. It is easy to see this in some simple cases, as in the following examples.

Examples.

If we want to compute $\sum_{x,y} f(x,z)g(y,z)$, we can compute this more efficiently as $(\sum_x f(x,z)) \left(\sum_y g(y,z)\right)$.

Another marginalization, $\sum_{x,z} f(x,z)g(y,z)$, can be computed as $\sum_z g(y,z) (\sum_x f(x,z))$.

While this reorganization is easy to do in simple cases, it may not always be obvious in more complicated cases. The junction tree formalism gives us a way to graphically indicate the computation making it easier to see how this may be done.

Definition. A *junction tree* is a tree in which each vertex is labeled by a finite set such that the intersection of the labels on any two vertices is a subset of the label of any vertex lying on the unique path between the two vertices. Equivalently the condition is that the subgraph induced by the vertices whose labels contain any given element is connected.

When there is a junction tree with the vertex labels being given by the sets S_i, then we can find the marginals β_i using the message passing algorithm described below.

Let the junction tree have M vertices and for each $i = 1, \ldots, M$, let vertex i be labeled with S_i. The message from vertex i to an adjacent vertex j is a function $\mu_{i,j} : A_{S_i \cap S_j} \to R$, i.e., it is a function of the variables $x_{S_i \cap S_j}$. It is computed using the local kernel α_i and the messages to vertex i from all its neighbours except vertex j by multiplying all these incoming messages with the local kernel and then marginalizing out all the variables not in S_j, i.e.,

$$\mu_{i,j}(x_{S_i \cap S_j}) = \sum_{x_{S_i \setminus S_j} \in A_{S_i \setminus S_j}} \alpha_i(x_{S_i}) \prod_{\substack{v_k \,\mathrm{adj}\, v_i \\ k \neq j}} \mu_{k,i}(x_{S_k \cap S_i}). \tag{2.4}$$

When vertex i has received messages from all its neighbours, it can compute a "result" $\sigma_i : A_{S_i} \to R$ as

$$\sigma_i(x_{S_i}) = \alpha_i(x_{S_i}) \prod_{v_k \,\mathrm{adj}\, v_i} \mu_{k,i}(x_{S_k \cap S_i}). \tag{2.5}$$

This "result" σ_i is the required marginalization β_i.

We need to consider the issue of scheduling the messages. The message passing rule implies that the message from vertex i to vertex j can be sent only when vertex i has received messages from all its other neighbours. So the messages have to be scheduled in such a way that these dependencies are satisfied for each message. Since initially no messages have been received, message passing begins from the leaves (who can send messages to their unique neighbour). Since removing the leaves still leaves a tree, it follows that we can continue to pass some more messages (since a tree has at least two leaves.) This can be continued until we are left with only one vertex. This vertex has now received messages from all its neighbours and so can send messages to all its neighbours. They in turn can send messages to their other neighbours. In this way, messages propagate out towards the leaves. The scheduling ends when a message has been passed on each edge along each direction. The above argument demonstrates that this can be achieved without deadlock. At this point it is possible to compute the result at each vertex.

If only one of the β_i needs to be computed, then we make vertex i the root to obtain a rooted tree, and then schedule only the messages that are in the direction pointing towards the root. Here too we start from the leaves and pass messages upward towards the root. Once the root has received messages from all its neighbours, the β_i can be computed.

We can make the following argument to see that the result of the message passing algorithm is the required marginalization. Let v and w be adjacent vertices. Removing the edge between v and w breaks the junction tree into two components. Let C_v denote the set of vertices in the component containing v and C_w, the set of vertices in the component containing w. Since the edge between v and w is on the unique path between any vertex in C_v and any vertex in C_w, we can see that any variable that occurs in some vertex in both C_v and C_w must occur in both the vertices incident on this edge. Thus the message passed on this edge is a function of exactly those variables that appear in both the components.

We claim that the message from v to w, which may be viewed as a message from C_v to C_w, is the product of all the local kernels of vertices in C_v marginalized over all the variables that do not occur in w. This is proved by induction on the number of vertices in C_v. The claim is easily seen to be true for a message from a leaf, i.e., $|C_v| = 1$. When $|C_v| \geq 1$, the outgoing message from v is the product of the local kernel and all but one of the incoming messages marginalized over the variables not in w. Each of the incoming messages are, by induction, products of sets of local kernels marginalized over all the variables not in v. The proof of the claim for the message from v to w follows from the fact that any variable occurring in more than one of the sets of local kernels forming the incoming messages will, by the junction tree property, also be in the given vertex, and hence not be marginalized out at an earlier stage.

The result computed at a vertex v is the product of its local kernel and all its incoming messages. By the claim, each of the incoming messages is the product of a set of local kernels marginalized over all the variables not occurring in v. Any variable occurring in more than one of these sets will, by the junction tree property, also be in v, and so will not be marginalized. Thus the result computed at v is the product of all the local kernels marginalized over all the variables that are not in v.

Thus, the junction tree condition ensures that a variable is summed over, in the process of computing a message, only when all the local kernels containing that variable have been multiplied into the product. It is allowable to take any other local kernel, i.e., any kernel not containing the given variable, out of this sum by distributivity so they need not have been multiplied in when the variable is summed over.

Example.

Let us consider the probabilistic state machine example given earlier. From the factorization of the joint probability in equation 2.3, we can form a junction tree for this in the following way. Since we want to find the conditional probabilities of each u_t, we have local domains $\{u_t\}$ with local kernels $p(u_t)$ for each $0 \leq t \leq n-1$. We also have the local domains $\{s_t, u_t, s_{t+1}\}$ for each $0 \leq t \leq n-2$ with associated local kernel $p(s_{t+1}|s_t, u_t)p(y_t|s_t, u_t)$ for $1 \leq t \leq n-2$ and kernel $p(s_0)p(s_1|s_0, u_0)p(y_0|s_0, u_0)$ for $t = 0$ (y_t is observed and thus fixed), and $\{s_{n-1}, u_{n-1}\}$ with kernel $p(y_{n-1}|s_{n-1}, u_{n-1})$. These can be made into a junction tree as shown in Figure 2.2 for the case $n = 4$.

An example of a scheduling order for message passing in this graph satisfying the dependency requirements is the following. First the leaves pass messages $1 \to 5$, $2 \to 6$, $3 \to 7$, and $4 \to 8$ inwards. Then we can pass the messages $5 \to 6$ and $8 \to 7$. Then we can pass $6 \to 7$ and $7 \to 6$. Now 6 and 7 have received all the incoming messages and can send out messages $6 \to 5$, $6 \to 2$, $7 \to 8$, and $7 \to 3$. Then 5 and 8 can send messages $5 \to 1$ and $8 \to 4$. Now all the vertices have received all incoming

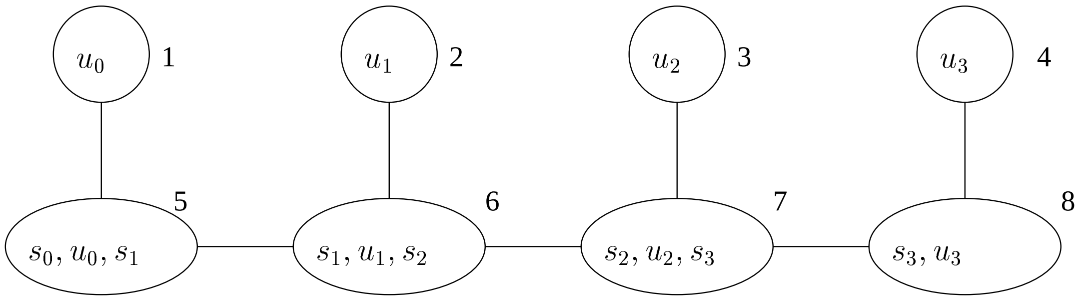

Figure 2.2: Junction tree for a probabilistic state machine/convolutional code.

messages. In this scheduling, each message was passed at the earliest time possible.

2.3 Complexity of the Message Passing Algorithm

Let us consider a vertex v with label $S(v)$ and degree $d(v)$. Then a message computation involves multiplying the $d(v)-1$ incoming messages into the local function, which will take $(d(v)-1)\left|A_{S(v)}\right|$ multiplications, and then marginalizing the table of $\left|A_{S(v)}\right|$ elements thus obtained to the variables that also are in the label of the target vertex (say v'), which will take $\left|A_{S(v)}\right| - \left|A_{S(v)\cap S(v')}\right|$ additions. We may bound the sum of these two above by $d(v)\left|A_{S(v)}\right|$, which gives us a bound on the number of arithmetic operations required for a message computation. A result computation at vertex v takes $d(v)\left|A_{S(v)}\right|$ multiplications, since all the incoming messages must be multiplied into the local function.

A single vertex result computation requires all but the target vertex to pass one message each, resulting in one message on each edge, and the target vertex to compute its result. The number of arithmetic operations this takes is at most

$$\sum_v d(v)\left|A_{S(v)}\right|, \tag{2.6}$$

and accounting for the $\left|A_{S(v)}\right| - \left|A_{S(v)\cap S(v')}\right|$ term exactly gives

$$\sum_v d(v)\left|A_{S(v)}\right| - \sum_{\substack{v_1, v_2 \\ v_1 \text{ adj } v_2}} \left|A_{S(v_1)\cap S(v_2)}\right| \tag{2.7}$$

arithmetic operations.

If we want to compute the result at all the vertices, and this is done in the obvious way, we get terms of the form $d(v)^2 |A_{S(v)}|$ for each vertex, since each vertex must send out $d(v)$ messages. But we may observe that it is possible to compute all the n possible products of $n-1$ numbers out of a set of n numbers in no more than $3n-6$ multiplications. Let $a_1, \ldots, a_n$ be the numbers. Let $b_i = \prod_{j=1}^{i} a_i$ for $i = 1, \ldots, n-1$. All the b_i may be computed, each b_i using the value of b_{i-1}, using $n-2$ multiplications. Similarly, $c_i = \prod_{j=i}^{n} a_i$ for $i = 2, \ldots, n$ may also be computed using $n-2$ multiplications. Then we can write

$$\prod_{j \neq i} a_j = b_{i-1} c_{i+1}, \tag{2.8}$$

where $b_0 = c_{n+1} = 1$ and so all such products can be computed using $n-2$ more multiplications, giving us a total of $3n-6$ multiplications. We observe that with one more multiplication, we can also compute b_n, which is the product of all the n terms. The all vertex computation requires each vertex to pass one message to each of its neighbours. For this, out of the $d(v)+1$ terms, which are the incoming messages and the local function, vertex v has to compute the $d(v)$ products of the $d(v)+1$ terms, each of which leaves out one incoming message. For its own result computation, it also needs the product of all the $d(v)+1$ terms. From the above argument, all this can be done with no more than $3d(v)$ multiplications. This has to be done for each of the $|A_{S(v)}|$ values of the variables the local function depends on, giving us a figure of $3d(v)|A_{S(v)}|$ multiplications as an upper bound. The marginalizations performed during the message computations remain as in the earlier case, and, summed over all the messages, this gives $\sum_v d(v)|A_{S(v)}| - 2\sum_{\substack{v_1, v_2 \\ v_1 \text{ adj } v_2}} |A_{S(v_1) \cap S(v_2)}|$ additions. So the all vertex computation takes no more than

$$4 \sum_v d(v) |A_{S(v)}| \tag{2.9}$$

arithmetic operations.

An upper bound on the complexity of a similar algorithm has been obtained in [18], as $3\sum_v q_v + M \max_v q_v$. This is strictly greater than the bound in equation 2.7.

2.4 Construction of Junction Trees

Given a junction tree with the S_i as vertex labels, we have a message passing algorithm to compute the marginals of the global function. There remains the question of whether, when we are given the sets S_i, there exists a junction tree with these sets as labels, and what can be done otherwise.

The question of whether such a junction tree exists is easy to answer. Suppose there is such a tree with vertices labeled by the S_i. Label each edge by the intersection of labels of the incident vertices. If a given variable x_j appears in l of the sets S_i, the number of times it appears on the edges is l minus the number of components of the subgraph induced by the vertices whose labels contain x_j. This can be at most $l-1$, which happens when this induced subgraph is connected. The tree we have is a junction tree if and only if the induced subgraph thus obtained for each variable x_j is connected, which means that the sum of the cardinalities of edge labels takes on its maximum possible value of $\sum_{i=1}^{M} |S_i| - n$. (n is the number of variables.) This gives us an algorithm to determine whether we can form a junction tree. We start out with a complete graph with the vertices labeled by the sets S_i. We give each edge a weight, which is the cardinality of the intersection of the labels of the two incident vertices. Then we find the maximum weight spanning tree, which can be efficiently done using Prim's algorithm [22]. Comparing the weight of this spanning tree with the above value it should have if it is a junction tree, we can tell whether or not we have a junction tree with the given vertex labels, with the maximum weight spanning tree being such a junction tree if one exists.

If no junction tree exists with the given vertex labels S_i, we can still find a junction tree such that each S_i is a subset of some vertex label, so that each local function α_i may be associated with a vertex whose label contains S_i. This can be done in a systematic way by forming a moral graph, triangulating it, and forming a junction tree of the cliques of the resulting graph, as described in [15, Section 4.5]. We describe this construction further and use it in section 3.2

In any case, the problem of finding a minimum complexity junction tree to solve a marginalization problem for given sets S_i has been proven to be NP-hard [16]. Thus we cannot hope to construct minimum complexity junction trees for all our marginalization problems, but we can often still construct junction trees that give us algorithms of reasonable complexity even if it is not minimal.

As an indication of the hardness of finding minimum complexity junction trees, we may see from the following example that even in the case where there is a junction tree with a given set of vertex labels, it may be possible to find a junction tree with additional vertices which can be used to solve the same marginalization problem with lower complexity.

Example.

We give an example to show that there are cases where a set of local domains can form a junction tree, but it is possible to obtain a lower complexity junction tree by adding more local domains. This illustrates the problem of finding the minimum complexity junction tree for a given marginalization problem.

Consider the local domains $\{x_1, x_2, x_3\}$, $\{x_1, x_4\}$, $\{x_2, x_5\}$, and $\{x_3, x_6\}$. There is a unique junction tree with these sets as vertex labels, given in Figure 2.3(a). Let the alphabet size for x_1 be equal to Q, and let all the other alphabet sizes be equal to q. Let us assume that Q is much bigger than q. Then, the complexity is dominated by the terms relating to vertices whose labels contain x_1. By equation 2.7, we can see that Q appears with highest power equal to one, and the linear term has coefficient $q + 3q^2 - 1$. If we add in the local domain $\{x_2, x_3, x_5\}$, then we can form the junction tree shown in Figure 2.3(b), in which the vertex labeled by $\{x_1, x_2, x_3\}$ has a lower degree and so the linear term in Q has coefficient $q+2q^2-1$. Thus the second junction tree gives a lower complexity than the first if Q is sufficiently large compared to q.

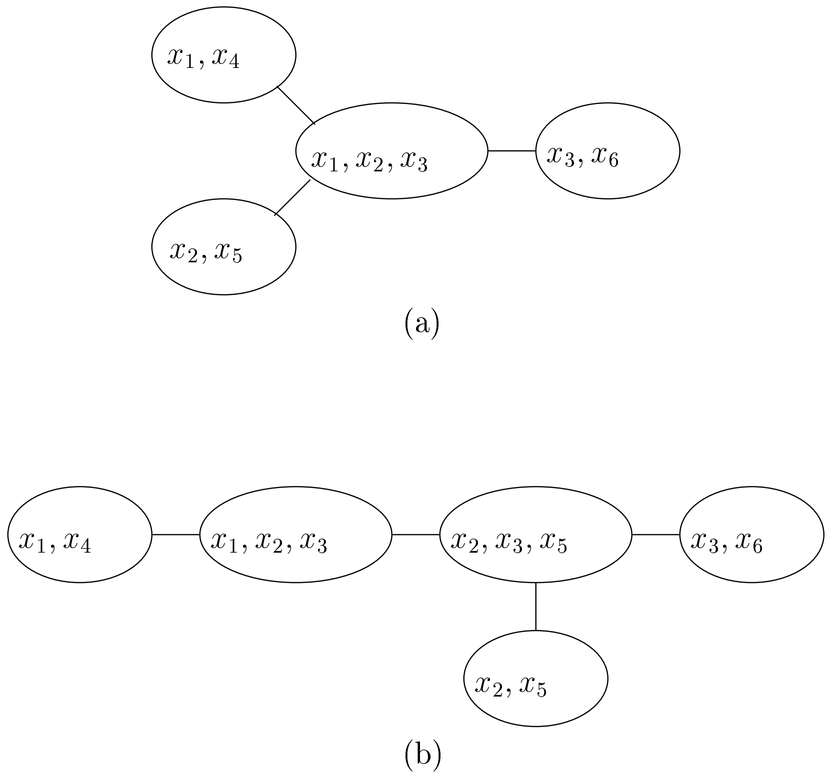

Figure 2.3: Junction trees for example where adding a local domain lowers complexity.

Chapter 3 The Discrete Fourier Transform

We attempt to find a graphical algorithm of the kind discussed in the previous chapter to compute the discrete Fourier transform (DFT). This is motivated by the fact that the fast Fourier transform [25] owes its efficiency to reorganizing computations using the distributive law. A junction tree is constructed using the method described in [15]. On obtaining a junction tree based algorithm, we find that we do recover the fast Fourier transform.

3.1 The Problem

For a positive integer N, the discrete Fourier transform of a function $f : \mathbb{Z}_N \to \mathbb{C}$ is to be another function $F : \mathbb{Z}_N \to \mathbb{C}$, given by

$$F(y) = \sum_{x=0}^{N-1} f(x) e^{i2\pi \frac{xy}{N}}. \tag{3.1}$$

We consider the case where $N = p^m$ for some prime p. Let $x = \sum_{i=0}^{m-1} x_i p^i$ and $y = \sum_{i=0}^{m-1} y_i p^i$ where the x_i and y_i belong to $\{0, \ldots, p-1\}$. Then the product xy becomes $\sum_{0 \le k,l < m} x_k y_l p^{k+l}$, and the exponential in equation 3.1 factorizes. We can then write

$$F(y_0, \ldots, y_{m-1}) = \sum_{x_0, \ldots, x_{m-1}} f(x_0, \ldots, x_{m-1}) \prod_{0 \le k+l < m} e^{i2\pi \frac{x_k y_l}{p^{m-k-l}}}, \tag{3.2}$$

the exponential factors being unity when $k + l \ge m$. This expresses F as a sum over the variables x_i of a product of many factors and is thus in the form of the marginalization problem in chapter 2.

The sets of variables on which the factors depend (i.e., the local domains) are $S_I = \{x_0, \ldots, x_{m-1}\}$ and $S_{k,l} = \{x_k, y_l\}$ for each k, l such that $0 \leq k+l < m$. We also add a set $S_F = \{y_0, \ldots, y_{m-1}\}$ for the vertex where the result F is obtained. It is not possible to find a junction tree with the above sets as vertex labels. So we instead need to construct a junction tree such that each of the above sets is a subset of some vertex label.

3.2 Moral Graphs and Triangulation

Given a set of local domains, we define the *moral graph* [15] to be a graph with the vertex set being the set of variables and having an edge between any two variables if there is some local domain with both these as elements.

Given a cycle in a graph, a chord is an edge between two vertices on the cycle that do not appear consecutively in the cycle. A graph is *triangulated* if every simple cycle (i.e., no repeated vertices) of length bigger than three has a chord.

In [15], it is shown that the cliques (maximal complete subgraphs) of a graph can be the vertex labels of a junction tree if and only if the graph is triangulated. Thus, to form a junction tree with vertex labels such that each of our local domains S_i is contained in some vertex label, we form the moral graph associated with the local domains S_i, add enough edges to the moral graph so that the resulting graph is triangulated, and then form a junction tree with the cliques of this graph as vertex labels.

We apply this method to the discrete Fourier transform problem. The moral graph has vertices $x_0, \ldots, x_{m-1}, y_0, \ldots, y_{m-1}$. There are edges between any two of the x_i (due to S_I), any two of the y_i (due to S_F), and between any x_k and y_l for which $k + l < m$ (due to $S_{k,l}$). Figure 3.1 shows this moral graph for $m = 4$.

The following argument shows that the moral graph constructed above is triangulated. The fact that all the x_i are adjacent and all the y_i are adjacent implies that in any cycle without a chord, any two x_i must occur next to each other and similarly with any two y_i. This restricts the length of such a cycle to four and the

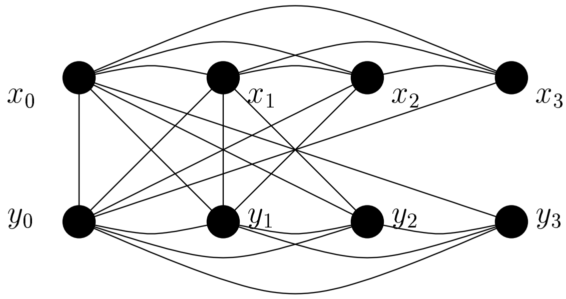

Figure 3.1: Moral graph for the DFT problem for $m = 4$.

cycle should be of the form x_a, x_b, y_c, y_d. If there is cycle of this form, then $a+d < m$ and $b + c < m$ since there are edges between x_a and y_d, and x_b and y_c. Thus $(a + c) + (b + d) = a + b + c + d < 2m$ which means that at least one of $a + c$ and $b + d$ is less than m. So there is an edge between x_a and y_c or an edge between x_b and y_d, giving us a chord for such cycles also.

Since the moral graph is triangulated, we do not add any more edges but find the cliques instead. The cliques have vertex sets $S'_i = \{x_0, \ldots, x_{m-i}, y_0, \ldots, y_{i-1}\}$, for $i \in \{1, \ldots, m\}$. The junction tree formed with these as vertex labels is a linear chain, where vertex i (with label S'_i) is adjacent to vertex j iff $j = i \pm 1$. We add to this graph a vertex 0 with label $S'_0 = S_I$, which is adjacent to vertex 1, and a vertex $m+1$ with label $S'_{m+1} = S_F$, adjacent to vertex m. This gives us a junction tree, as shown in Figure 3.2, where each of the original local domains is contained in some vertex label, which is what we set out to construct.

3.3 The Algorithm

For the junction tree we have constructed, the local function at each vertex i must be chosen to be some product of the factors we started out with, which are all functions of the variables which belong to the label S'_i. We let $\alpha_0 = f$ be the local function at vertex 0. For each k, l, $0 \leq k + l < m$, we associate the exponential $e^{i2\pi \frac{x_k y_l}{p^{m-k-l}}}$ with some vertex containing it. For i from 1 to m, we let α_i, the local function at vertex i, be the product of the exponentials associated with vertex i. Finally we let $\alpha_{m+1} = 1$

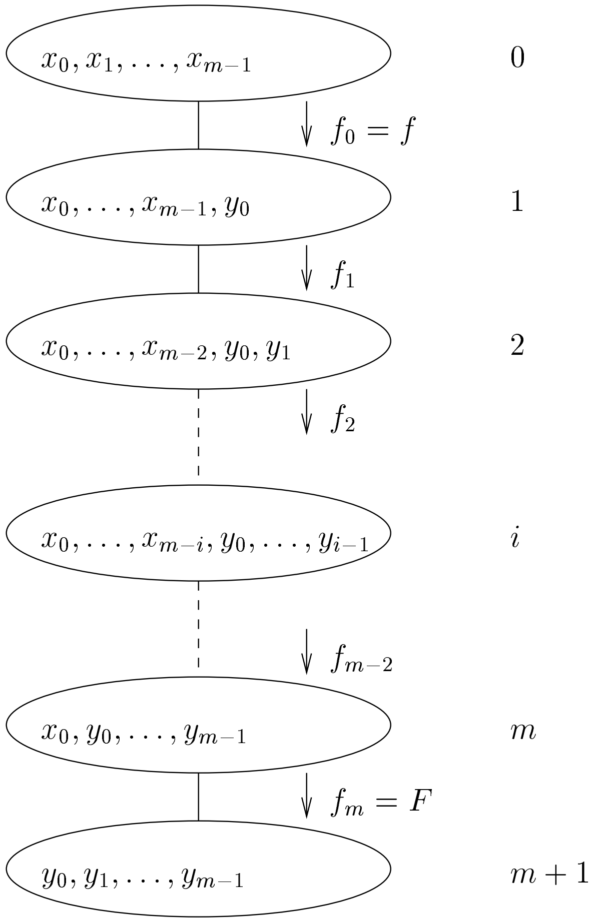

Figure 3.2: Junction tree for the discrete Fourier transform.

be the local function at vertex $m+1$. Now we can run the message passing algorithm to compute the marginalization of products β_{m+1} which is just the required Fourier transform F.

The message passing algorithm begins by passing a message from vertex 0 to vertex 1. Then messages are passed successively along the chain. When vertex i receives a message from vertex $i-1$, it then sends a message to vertex $i+1$. The algorithm terminates when vertex $m+1$ receives a message from vertex m. This message is the Fourier transform F. Denoting the message passed from vertex i to $i+1$ by f_i (which is a function of the variables $S'_i \cap S'_{i+1}$), we may describe the computation performed by the message passing algorithm as follows.

$$\begin{aligned} f_0 &= f, \\ f_{i+1} &= \sum_{x_{m-i-1} \in Z_p} f_i \alpha_{i+1} \text{ for } 0 \le i < m, \\ F &= f_M. \end{aligned} \tag{3.3}$$

There is in general more than one vertex with which the exponentials $e^{i2\pi \frac{x_k y_l}{p^{m-k-l}}}$ may be associated, so that we have several possibilities for choosing the α_i. It turns out that the choices correspond to variants of the fast Fourier transform algorithm [25]. We consider two of these cases. Associating each of the above exponentials with vertex i for the largest possible i gives us

$$\alpha_i = e^{\frac{i2\pi}{p^M} x_{M-i} \sum_{l=0}^{i-1} y_l} \tag{3.4}$$

and is the "decimation in time" version of the FFT. Associating the exponentials with the j for smallest possible j gives

$$\alpha_i = e^{\frac{i2\pi}{p^M} y_{i-1} \sum_{l=0}^{M-i} x_l} \tag{3.5}$$

and is the "decimation in frequency" version of the FFT [25].

3.4 Complexity

For each i from 1 to m, vertex i has $m+1$ variables of size p, and so the vertex size is p^{m+1}. On each edge, the message is a function of m variables, and so the messages are of size p^m. Counting the number of operations needed to compute the result the single vertex $m+1$, we find that it takes $p^{m+1}m = pN\log_p N$ multiplications (where $N = p^m$) and $(p-1)p^m m = (p-1)N\log_p N$ additions, which matches the $N\log N$ complexity achieved by the fast Fourier transform.

3.5 Discrete Fourier Transform on a Finite Abelian Group

A general finite abelian group G is the direct product of the form $G = \prod_{j=1}^{k} \mathbb{Z}_{p_j^{m_j}}$ [14]. It is possible to consider the discrete Fourier transform on such groups, which can be expressed as

$$F(y_{(1)},\ldots,y_{(k)}) = \sum_{x_{(1)}\in\mathbb{Z}_{p_1^{m_1}},\ldots,x_{(k)}\in\mathbb{Z}_{p_k^{m_k}}} f(x_{(1)},\ldots,x_{(k)}) \prod_{j=1}^{k} e^{i2\pi\frac{x_{(j)}y_{(j)}}{p_j^{m_j}}}. \tag{3.6}$$

The $x_{(j)}$ and $y_{(j)}$ can be written in digits base p_j and then the exponentials can be factored as in equation 3.2. We can construct a moral graph for this problem, but it is not triangulated, unless $k=1$, which is discussed earlier in this chapter. So, we need to add more edges to get a triangulated graph. It is possible to add edges in a way that gives us a junction tree corresponding to a fast Fourier transform on this product group.

Note that in the case of the discrete Fourier transform on $\mathbb{Z}_{p^m}$ we have a junction tree that starts with set $\{x_0,\ldots,x_{m-1},y_0\}$ and then, in each step, replaces the highest index x (x_{m-j}) by the next index y (y_j). We obtain a linear junction tree with these sets. In the case of the DFT on a product group, this needs to be done for each of the factors. So we start with $\{x_{(1),0},\ldots,x_{(1),m_1-1},y_{(1),0},\ldots,x_{(k),0},\ldots,x_{(k),m_k-1}\},y_{(k),0}$ and then at each step replace a highest index $x_{(l)}$, i.e., $x_{(l),m_l-j}$, by the next $y_{(l)}$, i.e.,

$y_{(l),j}$. Since at each step, we can choose any of the $x_{(l)}$ for this (except those l for which we are left with only $x_{(l),0}$), this gives us several linear junction trees with the same complexity.

While we can come up with these junction trees, since we are looking for something like the fast Fourier transform, we do not have an answer to the question of whether there are junction trees that give us even lower complexity.

Chapter 4 Single Cycle Junction Graphs

The behaviour of the message algorithm on general junction graphs with loops, discussed in section 4.1, is not theoretically understood, and in general seems quite complex. But in the case of junction graphs with a single cycle, we find that the behaviour of message passing algorithm can be understood in terms of repeated matrix multiplications. In the $(\mathbb{R}^+, +, \cdot)$ semiring, we can use the Perron-Frobenius theorem to obtain facts about the convergence and limiting values of the messages. This treatment is independent of but very similar to the analysis of message passing on single cycle graphs in [30], which also has an analysis on min-sum message passing.

4.1 Junction Graphs and the Message Passing Algorithm

Definition. A *junction graph* is a graph whose vertices and edges are labeled by sets, where each edge label is contained in the label of both the vertices on which the edge is incident, and such that the graph formed by taking only the vertices and edges whose labels contain any given element is a tree.

A junction graph whose underlying graph is a tree is a junction tree. In that case, the junction condition implies that the edge label is the intersection of the labels of the two incident vertices.

If we have sets of variables S_i, functions $\alpha_i : A_{S_i} \to R$, and a junction graph whose vertices are labeled by the S_i, the message passing rule defined in section 2.2 can essentially still be applied, with the modification that all the variables not on the edge label are summed over, rather than those not in the target vertex. So the algorithm can be applied for junction graphs that are not junction trees. In that case, the algorithm is not well defined, since there is no way to schedule the messages taking care of dependencies as we could in the case of junction trees. But if the messages are initialized in some way, and then messages are scheduled in some way, this gives us an iterative algorithm. The messages propagate along the cycles of the graph updating the message at each iteration. There is no notion of termination but instead, we have a notion of convergence. If the messages converge, we may compute the result at a vertex as before.

The turbo decoding algorithm [7] and the decoding algorithm for Gallager's low density parity check codes [12] are expressible as message passing algorithms on junction graphs with cycles [23]. Their performance, as well as numerical simulations on some junction graphs with cycles (see chapter 5) indicate that the "results" obtained in some way approximate the marginalization of the global function that was computed exactly in the case of junction graphs.

4.2 Message Passing on Subtrees of Junction Graphs

Let G be a junction graph with vertices $\{1, \ldots, M\}$ with local domains S_i and local kernels $\alpha_i(x_{S_i})$. We consider a subtree H of the junction graph G, together with incoming messages $\mu_{I,k}(x_{U_k})$ to any vertex of this subtree from an edge not in the subtree. When messages are passed within this subtree, we would like to know what the outgoing messages $\mu_{O,k}(x_{U_k})$ from these vertices are. To this end, we construct another graph H'. We first take the subtree. For each of the incoming messages $\mu_{I,k}$ to a given vertex, we add a leaf vertex with local domain being U_k and add an edge with label U_k between this leaf vertex and the given vertex. The local kernel on the added leaf is $\mu_{I,k}$. The graph so constructed remains a tree. The incoming messages from the added leaf vertices are exactly the incoming messages to the subtree from the rest of the junction graph, so that the outgoing messages resulting from message passing in the subtree are now given by the messages toward the added leaves.

If the graph H' is a junction tree, then the message on any edge is just the product of all the local kernels of vertices on one side of the edge, appropriately marginalized.

So the outgoing message $\mu_{O,k}$ can be written as

$$\mu_{O,k}(x_{U_k}) = \sum_{x_{S_H \setminus U_k} \in A_{S_H \setminus U_k}} \left[\prod_{i \in V(H)} \alpha_i(x_{S_i}) \prod_{k' \neq k} \mu_{I,k'}(x_{U_{k'}}) \right], \tag{4.1}$$

where $S_H = \cup_{i \in V(H)} S_i$.

The graph H' is a junction tree if and only if H is, since edge labels are subsets of labels of adjacent vertices. But given that H is only a subtree of a junction graph with loops, it need not be a junction tree. In that case, we take each variable x_j and find the subgraph of H' formed by taking the vertices and edges whose labels contain x_j. If this subgraph is not connected, we label the occurrences of x_j in each component differently, by $x_{j,1}$, $x_{j,2}$, and so on. This makes the graph H' into a junction tree. This relabeling does not alter the behaviour of the message passing algorithm, which is local and sees the occurrences of the same variable in different components as different. So the outgoing messages can be expressed in terms of the modified H' by equation 4.1.

Example.

In Figure 4.1 we have a junction graph, as well as a subtree of this junction tree. The part containing x_1 has three components and x_1 has been relabeled differently in each component. Thus, if the kernel associated with vertex i is α_i, then after message passing on this subtree, we have

$$\mu_{O,1}(x_{1,1}) = \sum_{x_{1,2}, x_{1,3}, x_2} \alpha_5(x_2) \prod_{i=1}^{3} \alpha_i(x_{1,i}, x_2) \prod_{i=2}^{3} \mu_{I,i}(x_{1,i}), \tag{4.2}$$

and similar expressions for $\mu_{O,2}$ and $\mu_{O,3}$.

4.3 Message Passing on Single Cycle Junction Graphs

By a *single cycle graph*, we mean a connected graph with only one simple cycle. Such a graph consists of the one cycle, and the other vertices can be partitioned so that the

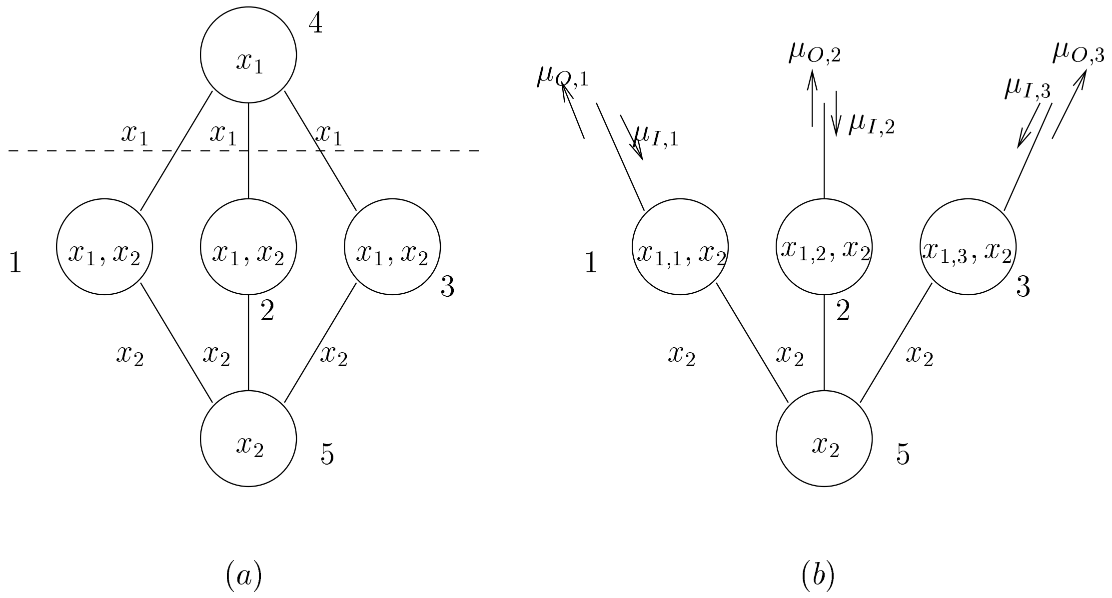

Figure 4.1: (a) A junction graph. (b) The subtree of (a) below the dashed line with the several occurrences of x_1 relabeled.

subgraph induced by each of the partitions is a tree that is attached in the original graph to one of the vertices on the cycle. Figure 4.2 is an example of a single cycle graph.

We note that messages passed inwards from the trees attached to the cycle are not altered by any further messages passed along the cycle or from the cycle into the tree. So we may take these messages to be initially computed and fixed while the other messages are passed. Fixing these messages to vertices on the cycle, the message from a vertex along an edge on the cycle depends only on the incoming message from the other edge on the cycle adjacent to this vertex. In particular, a message along the cycle in one direction has no dependence on a message in the other direction along the cycle. If, in each direction we pass messages around the cycle till the messages converge, if they will, and then pass messages from the cycle into the trees, we will have reached a fixed point of the message passing algorithm.

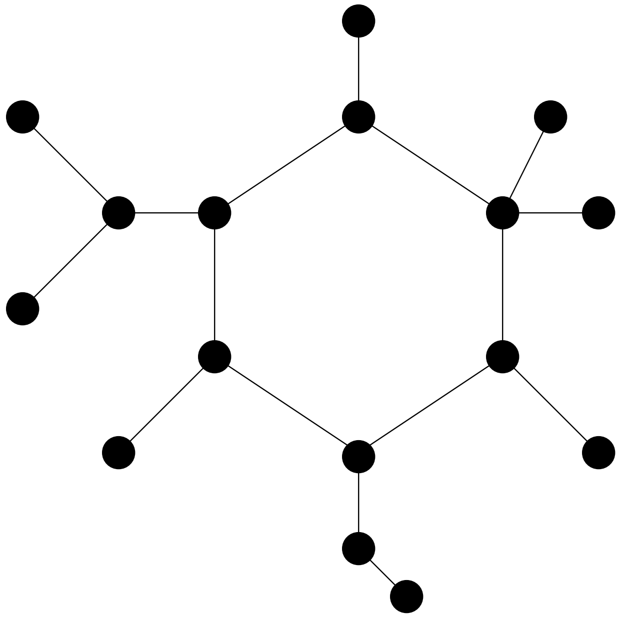

Figure 4.2: An example of a *single cycle graph.*

4.4 Analysis of Message Passing on Single Cycle Junction Graphs

We apply the results on message passing in subtrees of junction graphs (section 4.2) to analyzing the message passing algorithm on single cycle junction graphs. Let G now be a single junction cycle graph. We take H to be G with one of the edges on the cycle removed. Then H is a tree. Let the edge removed have label U and let the vertices incident to this edge be v and v'. Then the only incoming messages from the rest of G to H are the messages on the removed edge, which are incoming messages to v and v'. Let these be $\mu_{I,i}(x_U)$ for $i = 1, 2$. Now H is not a junction tree. The part of H containing any given variable not in x_U is connected, and the part of the graph containing any given variable in x_U has two components. So we relabel the variables that appear on the $\mu_{I,1}$ side as $x_{U,1}$ and the variables that appear on the $\mu_{I,2}$ side as $x_{U,2}$. With this relabeling, H is a junction tree which we can extend into the junction tree H'. Then using equation 4.1, we can compute the result of message passing for one cycle to get

$$\mu_{O,1}(x_{U,1}) = \sum_{x_{U^c}\in A_{U^c}, x_{U,2}\in A_U} \mu_{I,2}(x_{U,2}) \prod_{i\in V(G)} \alpha'_i(x_{S_i}) \tag{4.3}$$

and

$$\mu_{O,2}(x_{U,2}) = \sum_{x_{U^c}\in A_{U^c}, x_{U,1}\in A_U} \mu_{I,1}(x_{U,1}) \prod_{i\in V(G)} \alpha'_i(x_{S_i}), \tag{4.4}$$

where α'_i are the α_i with the appropriate variable relabeling done. If we define

$$\beta'(x_{U,1}, x_{U,2}) = \sum_{x_{U^c}\in A_{U^c}} \prod_{i\in V(G)} \alpha'_i(x_{S_i}), \tag{4.5}$$

we can then write

$$\mu_{O,2}(x_{U,2}) = \sum_{x_{U,1}\in A_U} \mu_{I,1}(x_{U,1})\beta'(x_{U,1}, x_{U,2}) \tag{4.6}$$

and

$$\mu_{O,1}(x_{U,1}) = \sum_{x_{U,2}\in A_U} \mu_{I,2}(x_{U,2})\beta'(x_{U,1}, x_{U,2}), \tag{4.7}$$

which describe multiplications by matrices with elements given by the function β'.

We also note that $\mu_{O,2}$ and $\mu_{O,1}$ are just going to be $\mu_{I,1}$ and $\mu_{I,2}$ respectively for the next round of message passing. So denoting $\mu_{I,1}$ and $\mu_{I,2}$ by μ_1 and μ_2 respectively and $\mu_{O,2}$ and $\mu_{O,1}$ by μ'_1 and μ'_2 respectively, and letting B be the matrix such that $B_{x_{U,1},x_{U,2}} = \beta'(x_{U,1}, x_{U,2})$, we get

$$\begin{aligned} \mu'_1 &= B^T\mu_1, \\ \mu'_2 &= B\mu_2. \end{aligned} \tag{4.8}$$

Thus we have expressed the effect of passing messages round the cycle once as multiplying the message in each direction by a matrix expressible in terms of the local kernels as given by equation 4.5.

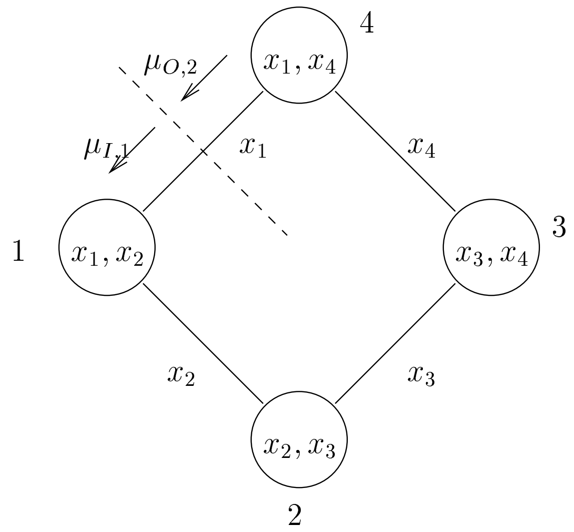

Figure 4.3: A single cycle junction graph.

Example.

Figure 4.3 shows a single cycle junction graph that can be made into a tree by breaking the edge along the dashed line. Then we can write down the result of message passing on this tree as

$$\mu_{O,2}(x_1) = \sum_{x_4} \alpha_4(x_1, x_4) \sum_{x_3} \alpha_3(x_3, x_4) \sum_{x_2} \alpha_2(x_2, x_3) \sum_{x_1} \alpha_1(x_1, x_2) \mu_{I,1}(x_1). \quad (4.9)$$

Let

$$\beta'(x_{1,1}, x_{1,2}) = \sum_{x_4} \alpha_4(x_{1,2}, x_4) \sum_{x_3} \alpha_3(x_3, x_4) \sum_{x_2} \alpha_2(x_2, x_3) \alpha_1(x_{1,1}, x_2), \quad (4.10)$$

so that we can then write

$$\mu_{O,2}(x_{1,2}) = \sum_{x_{1,1}} \mu_{I,1}(x_{1,1}) \beta'(x_{1,1}, x_{1,2}) \quad (4.11)$$

as in equation 4.6. A similar expression can be written for the messages in the other direction, as in equation 4.7.

4.5 Result of the Iterative Algorithm

The marginalization of the global function to the variables x_U which are on the edge is given by

$$\beta(x_U) = \sum_{x_{U^c} \in A_{U^c}} \prod_{i \in V(G)} \alpha_i(x_{S_i}). \tag{4.12}$$

We note that this is the same as the expression for $\beta'(x_{U,1}, x_{U,2})$ in equation 4.5, but without the relabeling of the x_U into $x_{U,1}$ and $x_{U,2}$. So we have

$$\beta(x_U) = \beta'(x_U, x_U), \tag{4.13}$$

which tells us the marginalization of the global kernel to x_U is given by the diagonal of the matrix B which we are iterating.

We now look at the "result" obtained by the iterative algorithm. By computing the result at either of the vertices to which the edge is incident and then marginalizing it to x_U, we can compute the result of the iterative algorithm $\gamma(x_U)$. By using the expression for outgoing messages in equation 2.4, this result, computed with either vertex, can be written as

$$\gamma(x_U) = \mu_1(x_U)\mu_2(x_U), \tag{4.14}$$

i.e., γ is the component-wise product of the vectors μ_1 and μ_2. If, on successive iteration by the matrices B^T and B, the messages μ_1 and μ_2 converge, then the result of the iterative algorithm may be taken to be the component-wise product of the limiting values of these messages.

4.6 The Sum-Product Algorithm

We now consider the case of the $(\mathbb{R}^+, +, \cdot)$ semiring, where the message passing algorithm is termed the "sum-product" algorithm. Now the matrix B is a matrix of real numbers. If a vector is repeatedly multiplied by B then, apart from an overall scalar factor which may be divided out to keep the numbers bounded, the vector converges to the eigenvector of B corresponding to the unique eigenvalue of largest modulus, if it exists. Then we call this the principal eigenvector of B. If there are multiple eigenvalues of largest modulus, we get oscillatory behaviour rather than convergence, and if there are multiple eigenvectors corresponding to the eigenvalue of largest modulus, we get dependence on initial conditions. This may be seen, for instance, by using the Jordan canonical form for B, by which we also see that B has a principal eigenvector iff B^T does.

In the $(\mathbb{R}^+, +, \cdot)$ semiring, the matrix B has nonnegative entries, and the Perron-Frobenius theorem [17] tells us that the largest eigenvalue of B will be real and positive and the principal eigenvectors of B and B^T have nonnegative entries.

4.6.1 Binary Valued Variables

When the edge we break has only one variable, and that variable is binary valued, we find that the matrix B is 2×2. In this case an explicit calculation, shown below, shows that the component-wise product of principal eigenvectors has the same ordering as the diagonal elements of the matrix. This means that, in the case of probabilistic inference, both the exact and iterative algorithms will choose the same value of the variable as the maximum likelihood value, and so the iterative algorithm gives the same answer as the exact algorithm for this case. This generalizes a similar result for a turbo code like structure in [24].

Calculation. Consider a 2×2 matrix B of nonnegative entries, given by

$$B = \begin{pmatrix} a & b \\ c & d \end{pmatrix}. \tag{4.15}$$

Without loss of generality, let $a \geq d$. Then, the larger eigenvalue is given by

$$\lambda_+ = \frac{a+d}{2} + \sqrt{\left(\frac{a-d}{2}\right)^2 + bc}. \tag{4.16}$$

The principal eigenvectors of B and B^T are $\begin{pmatrix} \lambda_+ - d \\ c \end{pmatrix}$ and $\begin{pmatrix} \lambda_+ - d \\ b \end{pmatrix}$ respectively. Our claim now amounts to $(\lambda_+ - d)^2 \geq bc$, which is easily verified, since

$$(\lambda_+ - d)^2 = bc + 2\left(\frac{a-d}{2}\right)^2 + (a-d)\sqrt{\left(\frac{a-d}{2}\right)^2 + bc}. \tag{4.17}$$

We find that for all bigger matrices there are cases where the exact and iterative algorithms do not pick out the same element as most probable, though it seems from the results of numerical experiments [1] that the iterative algorithm still gives the same answer (for most probable value) as the exact algorithm a reasonably large fraction of the time.

Results on the performance of the "min-sum" algorithm, i.e., when the semiring used is $(\mathbb{R} \cup \{\infty\}, \min, +)$, are given in [30], [10] and [2]. More recent results on min-sum message passing in general loopy graphs are in [31] and [32].

Chapter 5 Numerical Simulations

5.1 A Junction Graph for "Turbo" Codes

In chapter 2, Figure 2.2, we have seen how we can form a junction tree for a convolutional code. If we ignore efficiency considerations, we can choose to ignore the structure of the code and make a junction tree consisting of local domains $\{u_i\}$ with local kernels $p(u_i)$, another local domain $\{u_1, \ldots, u_k\}$ with local kernel $p(Y|u_1, \ldots, u_k)$, as in Figure 5.1. This is a possible junction tree for any code, but doing the computation for the vertex labeled by $\{u_1, \ldots, u_k\}$ is too complex for large k. For a convolutional code, this computation may in fact be done efficiently because the local kernel $p(Y|u_1, \ldots, u_k)$ factorizes as in equation 2.3.

In a turbo code, the input bits are encoded by two encoders, with the input bits being permuted before being encoded by one of the encoders. Permuting the input bits before encoding only changes the local function $p(Y|u_1, \ldots, u_k)$ (by a permutation of the arguments) but not the structure of the junction tree in Figure 5.1. Since we

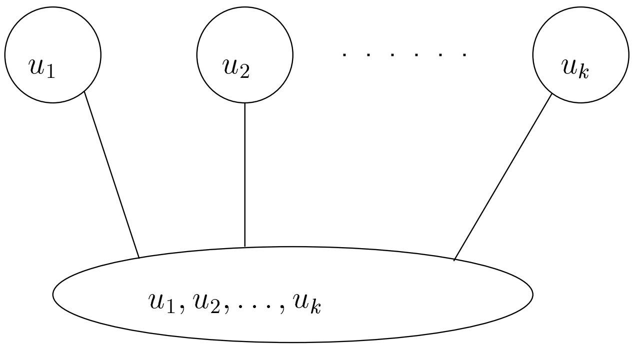

Figure 5.1: Simplified junction tree for decoding any code.

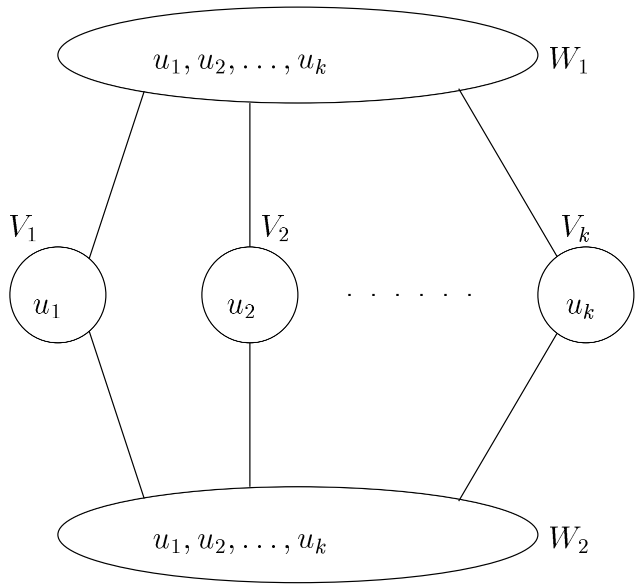

Figure 5.2: Simplified junction graph for "turbo" codes.

have two encoders, the function we want to marginalize is

$$p(Y_1, Y_2, u_1, \ldots, u_k) = \prod_{i=1}^{k} p(u_i) p(Y_1|u_1, \ldots, u_k) p(Y_2, u_1, \ldots, u_k). \tag{5.1}$$

This may be represented by a junction graph as shown in Figure 5.2.

The turbo decoding algorithm is equivalent to a message passing algorithm on this graph [23], though the vertices W_1 and W_2 are exploded into subtrees based on Figure 2.2, so that the computation is done more efficiently. The messages are initialized to constant functions, and then messages are alternately passed from W_1 to W_2 through the V_i and then back from W_2 to W_1. After this is iterated several times, the result is computed at vertices V_i.

5.2 Simulations of Small Turbo Like Junction Graphs

We consider the junction graph of Figure 5.2 with local kernel equal to one for vertices V_i and equal to $f_{1,2}(u_1, \ldots, u_k)$ for vertices $W_{1,2}$. The "result" we wish to compute is the marginalization of $f_1 \cdot f_2$, i.e.,

$$\beta_i(u_i) = \sum_{\{u_j\}_{j \neq i}} f_1(u_1, \ldots, u_k) f_2(u_1, \ldots, u_k). \tag{5.2}$$

We could also perform the iterative algorithm on the junction graph, as in the turbo decoding algorithm, and obtain "results" $\beta_i'(u_i)$ from this. For small k, it is feasible to compute both $\beta_i(u_i)$ and $\beta_i'(u_i)$, and compare the two after normalizing both of them.

We consider the junction graph of Figure 5.2 for small k, i.e., $k = 3$ and $k = 4$. Let the u_i be binary valued variables. We choose the functions f_1 and f_2 independently and uniformly on the space of 2^k nonnegative real numbers that sum to one (which forms a $2^k - 1$-simplex). Then we compute $\beta_i(u_i)$ and $\beta_i'(u_i)$, the latter being computed after the iteration is done a certain number of times. The result $\beta_i(u_i)$ is, after being normalized so that $\beta_i(0) + \beta_i(1) = 1$, specified completely by $\beta_i(0)$, and similarly for $\beta_i'(u_i)$. Thus we compare the two by plotting the point $(\beta_1(0), \beta_1'(0))$ for each choice of f_1 and f_2. (It does not matter which i we choose since the functions are picked from an ensemble that is symmetric under input bit permutation.)

Figure 5.3 shows such a plot for $k = 3$, with f_1 and f_2 being chosen 20000 times and the iteration for the β' computation being done 10 times. (Once from W_1 to W_2 and then from W_2 to W_1 counts as one iteration.) Figure 5.4 shows a similar plot for $k = 4$, also with 20000 points and 10 iterations for the β' computation.

The two algorithms, exact and iterative, result in the same decision if $\beta_i(0)$ and $\beta_i'(0)$ are on the same side of 0.5. We see from the plots that this happens a significant fraction of the time, and also that when they lie on opposite sides of 0.5, the exact result is itself fairly close to 0.5. This lends support to the belief [23] that there are undiscovered theorems about the performance of the iterative algorithm on junction graphs. In particular, it supports the conjecture that if the exact decision is fairly certain, i.e., not near 0.5, then the iterative result will with high probability be on the same side of 0.5 as the exact result.

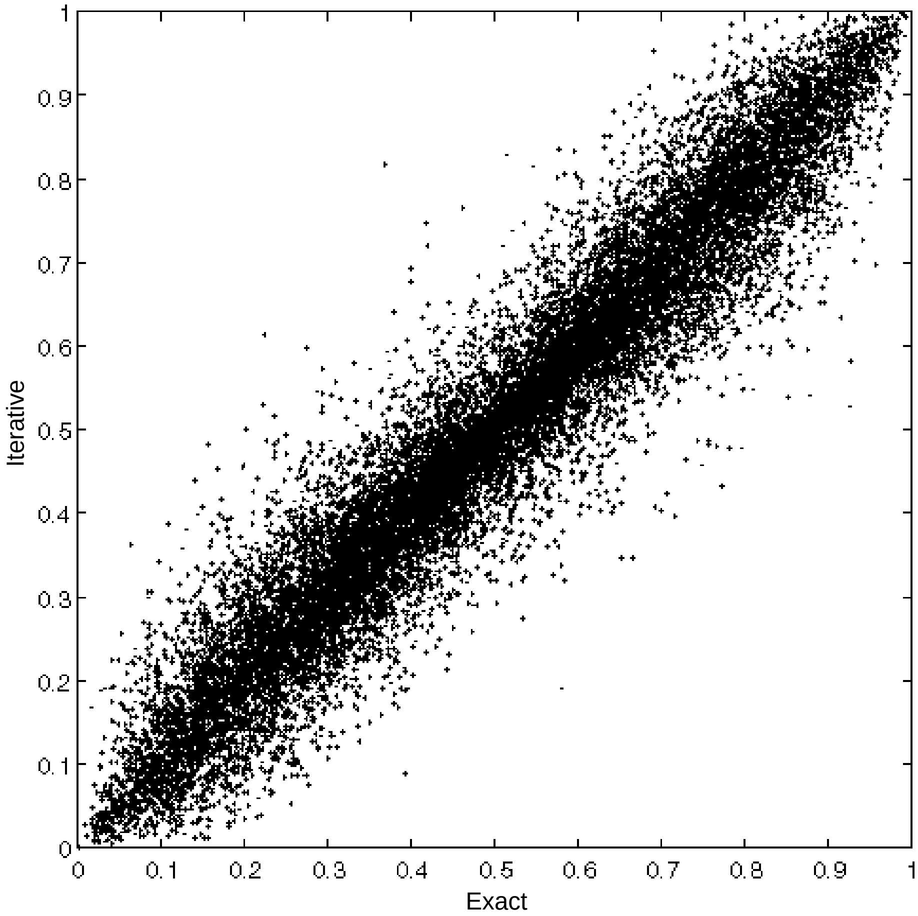

Figure 5.3: Comparison of exact and iterative results for a turbo like junction graph (Figure 5.2) for $k = 3$.

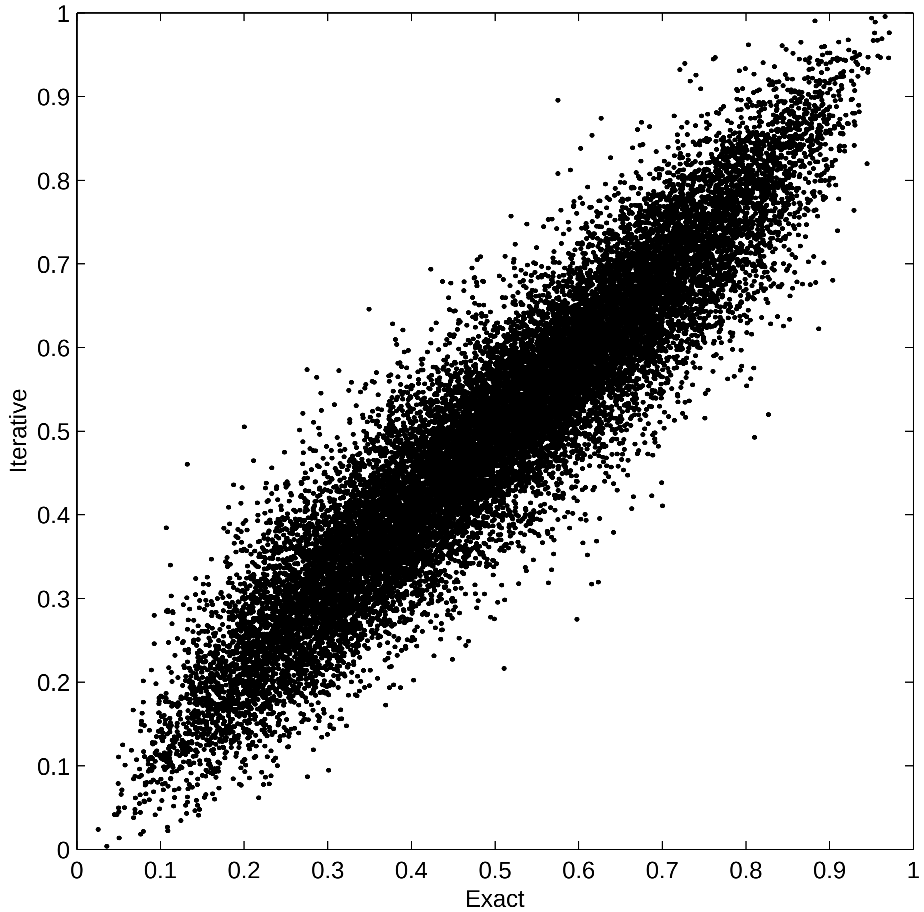

Figure 5.4: Comparison of exact and iterative results for a turbo like junction graph (Figure 5.2) for $k = 4$.

Chapter 6 Typical Set Decoding on the Gaussian Channel

The typical set decoding technique [21, 4] has been used to obtain noise thresholds for families of codes, such that for the noise level below the threshold it is possible to decode with probability of error going to zero as the block length is taken to infinity. Here we apply this technique to the Gaussian channel, which also gives us a new derivation of the noise threshold given by the "Simple" bound in [8].

6.1 Typical Set Decoding

Consider a code C. The *typical pairs* decoder [4] is one that when given a received sequence y, decodes to the unique $x \in C$ such that (x, y) belongs to the set of typical pairs T, if such an x exists. Let us define

$$T(x) = \{y|(x, y) \in T\},$$

so that $(x, y) \in T$ is equivalent to $y \in T(x)$. The probability of decoding error given that codeword x_0 is transmitted can be bounded as

$$P_e \leq P_I + P_{II}, \tag{6.1}$$

where

$$P_I = \Pr\{y \notin T(x_0)\}, \tag{6.2}$$

and

$$P_{II} = \sum_{\substack{x' \in C \\ x' \neq x_0}} \Pr\{y \in T(x_0) \cap T(x')\}. \tag{6.3}$$

We need some symmetry conditions to go further. We assume that the channel is a memoryless binary input symmetric channel [13], i.e., the channel inputs are the binary values 0 and 1 and the channel outputs are real numbers and we have

$$p(Y/0) = p(-Y/1), \tag{6.4}$$

for any output value Y. Consider the action on any channel input string x and channel output string y by a binary string u (all of the same length n) as follows. u acts on x by addition modulo two. u acts on y by negating all those components of y where u is 1. Equation (6.4) implies that the channel probability $p(y/x)$ is invariant under this action by any string u. If we also impose the condition that the typical pairs set T is invariant under this action, i.e., $(u(x), u(y)) \in T$ iff $(x, y) \in T$), then the P_e expression above becomes independent of the transmitted codeword x_0, which we will subsequently take to be the all zero codeword. If we also take the set T to be invariant under coordinate permutations (the channel is memoryless, so $p(y/x)$ is invariant under coordinate permutations), then we get $\Pr\{y \in T(x_0) \cap T(x')\}$ to depend only on the weight of the difference between x_0 and x'. Let $P_h(T) = \Pr\{y \in T(x_0) \cap T(x')\}$ when x_0 and x' differ in h positions.

Now, taking x_0 to be the all zero codeword, we can group the terms in P_{II} according to the weight of x' and write

$$P_{II} = \sum_{h=1}^{n} A_h P_h(T), \tag{6.5}$$

where A_h is the number of codewords of weight h. Note that this expression is linear in the weight enumerator, so the average P_{II} for an ensemble of codes of given block size can be got just by using the average weight enumerator $\overline{A}_h^{(n)}$ for these codes.

Now we want to use this bound for calculating a noise threshold for an ensemble of codes. We wish to define T such that P_I goes to zero as the blocksize goes to infinity. As in [4], one could define the typical pairs T_n for codes of block size n and for the BSC with crossover probability p as being all the pairs (x, y) where the fraction of positions in which they differ is within ϵ_n of p, where $\epsilon_n \longrightarrow 0$ more slowly than $n^{-1/2}$.

We also want to write $P_h(T_n)$ as

$$P_h(T_n) = e^{-n(K(\delta,s)+o(1))}, \tag{6.6}$$

where $\delta = h/n$, and s is a noise parameter (it could be the crossover probability p for BSC, or the noise variance σ^2 for Gaussian channel).

We have an ensemble of codes of varying block lengths [4]. Let $\overline{A}_h^{(n)}$ be the average weight enumerator for blocklength n. We have the ensemble spectral shape

$$r(\delta) = \lim_{n \longrightarrow \infty} \frac{1}{n} \overline{A}_{\lfloor \delta n \rfloor}^{(n)}, \text{ for } 0 < \delta \leq 1. \tag{6.7}$$

Then,

$$\overline{A}_h^{(n)} = e^{n(r(\delta)+o(1))}, \tag{6.8}$$

where again $\delta = h/n$.

From equations (6.5), (6.6), and (6.8), we can write P_{II} as

$$P_{II} = \sum_{\delta = \frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}, 1} e^{n(-K(\delta,s)+r(\delta)+o(1))}. \tag{6.9}$$

We define the typical set decoding threshold to be the supremum of values s for which

$$\sup_{0<\delta\leq 1} -K(\delta, s) + r(\delta) \leq 0. \tag{6.10}$$

Note that here, $K(\delta, s)$ depends only on the channel and $r(\delta)$ depends only on the code ensemble.

We want to show that whenever s is below this threshold, P_{II} goes to zero as $n \longrightarrow \infty$. If we did not have the $o(1)$ term in equation (6.9), we would have P_{II} going to zero as $n \longrightarrow \infty$ when $\sup_{0<\delta\leq 1} -K(\delta, s) + r(\delta) < 0$. With the $o(1)$ term, we can say only that the sum of all the terms for which δ is above any given positive value goes to zero. So the terms for very small δ need to be considered separately. This requires some conditions on the low weight codewords of the code ensemble. For an analysis of this and sufficient conditions on the code ensemble, see [4]. One sufficient condition is that the ensemble minimum distance of the code increase linearly with n.

Now we describe a way of choosing T for a binary input symmetric channel so that it satisfies the symmetry conditions. We do this by defining

$$T = \{(x,y)|x(y) \in T'\} \text{ for some } T', \tag{6.11}$$

where $x(y)$ is the result of x acting on y according to the action of a binary string on y given above. Now T is invariant under the action of any binary string u because

$$(u(x))(u(y)) = (x+u+u)(y) = x(y). \tag{6.12}$$

For T to be invariant under coordinate permutations, we must also have T' to be so. Note that T' is the set of typical outputs when the all zero string is transmitted.

We basically need to choose T' so that P_I does go to zero as the block length increases. Given this, the smaller we choose the set T', the smaller will be P_{II}, and potentially, we will have a better threshold.

6.2 Typical Set Decoding on the AWGN Channel

The additive white Gaussian noise (AWGN) channel is one where the binary inputs 0 and 1 are modulated into the real numbers $+1$ and -1 respectively and then a Gaussian random variable of variance σ^2 is added to this. i.e., we have

$$y = x(\mathbf{1}) + z, \tag{6.13}$$

where $\mathbf{1}$ is the sequence of n 1's and z is a sequence of i.i.d random variables that are Gaussian with mean zero and variance σ^2. For this channel, $s = \sigma^2$ is used as the measure of noise level.

Using the method in the previous section of specifying a T' to define the typical set T, the test for typicality of (x, y) becomes a test of $x(y)$. Since in this case, $x(y) = \mathbf{1} + x(z)$ and $x(z)$ is also sequence of i.i.d Gaussian random variables, we can equivalently make the test for typicality a test of $u = x(y) - \mathbf{1}$, which is a sequence of i.i.d random variables which are Gaussian with zero mean and variance σ^2. That is, we define T as all pairs (x, y) for which $u = x(y) - \mathbf{1}$ lies in some set U.

Different choices of U give us different $K(\delta, \sigma^2)$ and thus potentially different thresholds for any code. We consider some choices for U below.

6.2.1 Testing the Variance of the Noise

Our first choice for U is

$$U_1 = \{(u_1, \ldots, u_n) | \left| \frac{\sum u_i^2}{n} - \sigma^2 \right| < \epsilon\}, \tag{6.14}$$

for any small positive number ϵ. This set satisfies the condition that $P_I \longrightarrow 0$ since if x_0 is transmitted, and $u = x_0(y) - \mathbf{1}$, $\Pr\{u \notin U_1\} \longrightarrow 0$ as $n \longrightarrow \infty$, as can be shown using Chebyshev's inequality. For any (x, y), letting z be that noise value for which we have $y = x(\mathbf{1}) + z$, we have $u = x(z)$. Since the test for being in the set U is only on the sum of the squares, which is the same for u and z, the condition for typicality is that the hypothetical noise $z = y - x(\mathbf{1})$ have sample variance close to σ^2. So y is in $T(x)$ if it lies in a spherical shell centered at $x(\mathbf{1})$, having radius $\sigma\sqrt{n}$ and thickness $\sim \epsilon\sqrt{n}$.

Now we need to find $P_h(T) = \Pr\{y \in T(x_0) \cap T(x')\}$, where x_0 was transmitted and x' differs from x_0 in h positions. Since x_0 was transmitted, y is in the spherical shell around $x_0(\mathbf{1})$ with probability approaching 1 and since for AWGN, $p(y/x)$ is dependent only the distance between y and $x_0(\mathbf{1})$, we can take the probability of y to be distributed uniformly within that shell. So we can take $\Pr\{y \in T(x_0) \cap T(x')\}$ to be the ratio of the volume of $T(x_0) \cap T(x')$ to the volume of $T(x_0)$.

Figure 6.1 is a schematic of the sets $T(x_0)$ and $T(x')$. The radius of the spherical shells $T(x_0)$ and $T(x')$ is $r = \sigma\sqrt{n}$. The Euclidean distance d between $X_0 = x_0(\mathbf{1})$

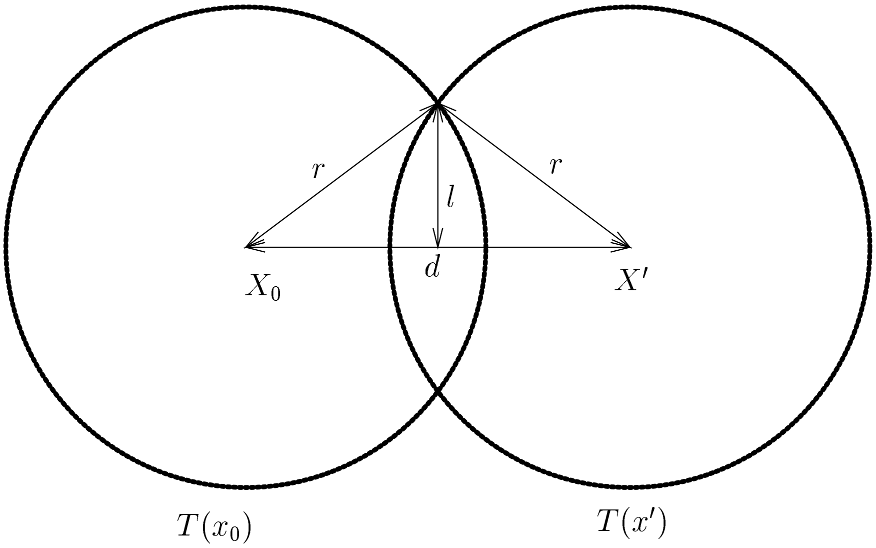

Figure 6.1: Spherical shell typical sets obtained by considering only the variance of the noise for typicality.

and $X' = x'(\mathbf{1})$ is given by $d = 2\sqrt{h}$. From Figure 6.1, we see that $T(x_0) \cap T(x')$ is a spherical shell of dimension $n-2$, with radius $l = \sqrt{r^2 - (\frac{d}{2})^2}$ and thickness $\sim \epsilon\sqrt{n}$ in the other two dimensions. Taking $\delta = h/n$, we have

$$l = \sigma\sqrt{n}\sqrt{1 - \delta/\sigma^2}. \tag{6.15}$$

The volume of a spherical shell of radius r and dimension k (in $\mathbb{R}^n$) is $A_k r^k$ times the thickness of the shell in the remaining $n-k$ dimensions, where A_k is the *volume* of the unit k dimensional sphere, equal to $2\pi^{(k-1)/2}/\Gamma((k-1)/2)$. (This is the perimeter of the 1-sphere in $\mathbb{R}^2$, the surface area of the 2-sphere in $\mathbb{R}^3$, and so on.) We only need the fact that A_{k+1}/A_k is bounded by a polynomial in k.

We get

$$\mathrm{Vol}(T(x_0)) \sim A_{n-1}(\sigma\sqrt{n})^{n-1} \cdot \epsilon\sqrt{n}, \tag{6.16}$$

and

$$\mathrm{Vol}(T(x_0) \cap T(x')) < A_{n-1}(\sigma\sqrt{n}\sqrt{1-\delta/\sigma^2})^{n-2} \cdot (\epsilon\sqrt{n})^2. \tag{6.17}$$

So, denoting $P_h(T) = P_{\delta n}(T)$ by P_δ,

$$P_\delta \sim< \frac{A_{n-2}}{A_{n-1}\sigma}(\sqrt{1-\delta/\sigma^2})^{n-2} \cdot \epsilon, \tag{6.18}$$

which gives us

$$\frac{\log P_\delta}{n} \longrightarrow \frac{1}{2}\log(1-\delta/\sigma^2), \tag{6.19}$$

as $n \longrightarrow \infty$. i.e.,

$$K(\delta, \sigma^2) = -\frac{1}{2}\log(1-\delta/\sigma^2). \tag{6.20}$$

To be on the good side of the threshold, we must have

$$\sup_{0<\delta\leq 1} \frac{1}{2}\log(1-\delta/\sigma^2) + r(\delta) < 0, \tag{6.21}$$

which is equivalent to

$$\frac{1}{2\sigma^2} > \sup_{0<\delta\leq 1} \frac{1-e^{-2r(\delta)}}{2\delta}. \tag{6.22}$$

The signal to noise ratio E_s/N_0 is $1/(2\sigma^2)$ and equation (6.22) tells us that the threshold on the signal to noise ratio for an ensemble of codes for achieving asymptotically zero probability of error as block length increases is $\sup_{0<\delta\leq 1} \frac{1-e^{-2r(\delta)}}{2\delta}$.

6.2.2 Testing the Variance and Mean

We now refine our test for typicality by taking the set U to be

$$U_2 = U_1 \cap \{(u_1, \ldots, u_n) | \left|\frac{\sum u_i}{n}\right| < \epsilon\}, \tag{6.23}$$

which is asking not only the sample variance to be close to the variance σ^2 but also the sample mean to be close to zero. U_2 is a spherical shell centered at zero of radius $\sigma\sqrt{n}$ of dimension $n-2$, restricted to the hyperplane perpendicular to the vector $\mathbf{1}$, with thickness $\sim \epsilon\sqrt{n}$ in the radial direction and thickness $\sim \epsilon\sqrt{n}$ in the direction perpendicular to the hyperplane. ($\left|\frac{\sum u_i}{n}\right| < \epsilon$ means that $u \cdot (\mathbf{1}/\sqrt{n}) < \epsilon\sqrt{n}$ and $\mathbf{1}/\sqrt{n}$ is the unit normal perpendicular to the hyperplane and so $\epsilon\sqrt{n}$ is the thickness

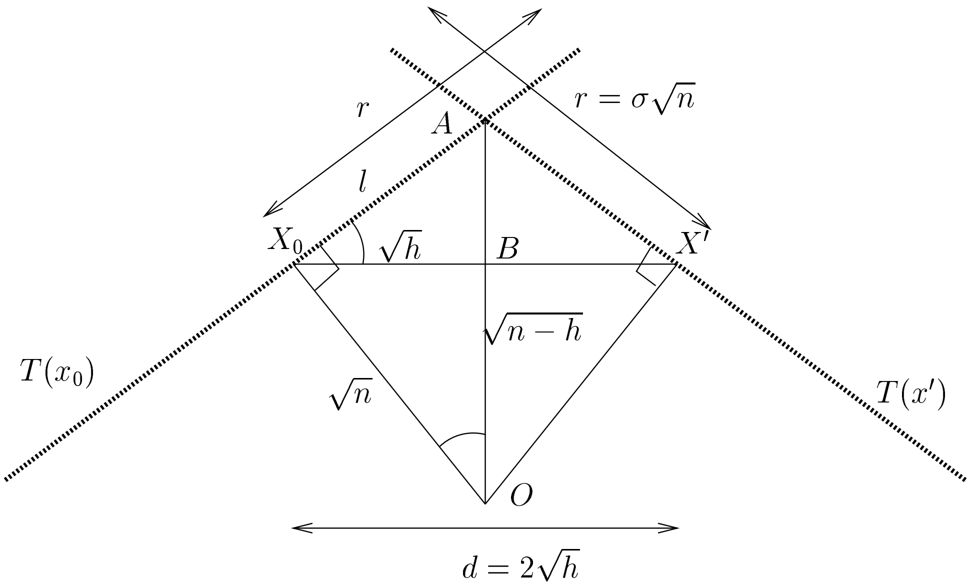

Figure 6.2: Intersection of two typical sets with typicality defined considering the variance and mean of the noise. This is a projection onto the plane containing the origin O, X_0 and X'.

perpendicular to the hyperplane.)

This set too satisfies $P_I \longrightarrow 0$ as $n \longrightarrow \infty$ since if x_0 was transmitted, $u = x_0(y) - \mathbf{1}$ is a sequence of i.i.d Gaussian random variables with mean zero and variance σ^2, and we have both $\left|\frac{\sum u_i^2}{n} - \sigma^2\right| < \epsilon$ and $\left|\frac{\sum u_i}{n}\right| < \epsilon$ with high probabilities approaching 1.

With this choice of U, $T(x_0)$ is a spherical shell of $n-2$ dimensions, centered at $x_0(\mathbf{1})$ with radius $\sigma\sqrt{n}$, and concentrated around the hyperplane passing through $x_0(\mathbf{1})$ and perpendicular to the vector $x_0(\mathbf{1})$ (with the same thickness in the other dimensions as U_2). Figure 6.2 is a projection of $T(x_0)$ and $T(x')$ onto the plane containing the origin, $x_0(\mathbf{1})$ and $x'(\mathbf{1})$. When x_0 and x' differ in h positions, the Euclidean distance between $X_0 = x_0(\mathbf{1})$ and $X' = x'(\mathbf{1})$ is $2\sqrt{h}$. In Figure 6.1, we see that $T(x_0) \cap T(x')$ is a spherical shell of dimension $n-4$, with radius $\sqrt{r^2 - l^2}$, where $r = \sigma\sqrt{n}$. From triangles OX_0B and X_0AB we have $l/\sqrt{h} = \sqrt{n}/\sqrt{n-h}$, which gives

$$l = \sqrt{n}\frac{\sqrt{h/n}}{\sqrt{1-h/n}} = \sqrt{n}\sqrt{\frac{\delta}{1-\delta}}, \tag{6.24}$$

where $\delta = h/n$. So the radius of the spherical shell $T(x_0) \cap T(x')$ (whose projection in the figure is just the point A) is

$$\sqrt{r^2 - l^2} = \sigma\sqrt{n}\sqrt{1 - \frac{\delta}{(1-\delta)\sigma^2}}, \tag{6.25}$$

and its volume is

$$\mathrm{Vol}(T(x_0) \cap T(x')) \sim< A_{n-4}\left(\sigma\sqrt{n}\sqrt{1 - \frac{\delta}{(1-\delta)\sigma^2}}\right)^{n-4} \cdot (\epsilon\sqrt{n})^4. \tag{6.26}$$

We also have

$$\mathrm{Vol}(T(x_0)) = A_{n-2}(\sigma\sqrt{n})^{n-2} \cdot (\epsilon\sqrt{n})^2. \tag{6.27}$$

So,

$$P_\delta = \frac{\mathrm{Vol}(T(x_0) \cap T(x'))}{\mathrm{Vol}(T(x_0))} = \frac{A_{n-4}}{A_{n-2}\sigma^2}\left(\sqrt{1 - \frac{\delta}{(1-\delta)\sigma^2}}\right)^{n-4} \cdot \epsilon^2 \tag{6.28}$$

and we get

$$\frac{\log P_\delta}{n} \longrightarrow \frac{1}{2}\log(1 - \frac{\delta}{(1-\delta)\sigma^2}), \tag{6.29}$$

as $n \longrightarrow \infty$. i.e.,

$$K(\delta, \sigma^2) = -\frac{1}{2}\log(1 - \frac{\delta}{(1-\delta)\sigma^2}). \tag{6.30}$$

The condition for being on the good side of the threshold is

$$\sup_{0<\delta\leq 1} \frac{1}{2}\log(1 - \frac{\delta}{(1-\delta)\sigma^2}) + r(\delta) < 0, \tag{6.31}$$

which, as in the previous case, can be transformed into

$$\frac{1}{2\sigma^2} > \sup_{0<\delta\leq 1} \frac{1 - e^{-2r(\delta)}}{2\delta}(1-\delta). \tag{6.32}$$

This tells us that the threshold on the signal to noise ratio $E_s/N_0 = 1/(2\sigma^2)$ for an ensemble of codes for achieving asymptotically zero probability of error as block length increases is $\sup_{0<\delta\leq 1} \frac{1-e^{-2r(\delta)}}{2\delta}(1-\delta)$.

Comparing with the thresholds in [8], we see that equation (6.22) gives us the threshold obtained by the Hughes bound and equation (6.32) gives us the (tighter) threshold obtained using any of several bounding techniques, including the "Simple" bound in that paper.

6.2.3 A Stronger Test for Typicality

It is possible to define a stronger notion of typicality than that of the earlier sections, and thus obtain better thresholds, as given in [5]. This uses a notion of typicality well defined on a channel with finite number of outputs and considers the Gaussian channel to be the limit of a sequence of such channels. For binary input symmetric channels whose outputs take on only a finite number of values, we define (x, y) to be typical only if the fraction of times a given output value Y_0 occurs in the sequence $x(y)$ is within a given ϵ of the probability $p(Y_0/0)$. This is testing not just the variance or mean but the distribution of values in $x(y)$. Taking a sequence of such typical set decoders which consider the Gaussian channel to be more and more finely quantized versions, we get the limiting $K(\delta)$ given by

$$K(\delta, \sigma^2) = H(\delta) - \max_{\int_0^\infty \delta(y)dy = \delta} \int_0^\infty G(\pi(y), f(y), \delta(y))dy, \tag{6.33}$$

where

$$G(\pi, f, \delta) = f\pi H(\frac{\delta}{2f\pi}) + (1-f)\pi H(\frac{\delta}{2(1-f)\pi}), \tag{6.34}$$

$$\pi(y) = (P(y/0) + P(y/1)) = \frac{1}{\sqrt{2\pi\sigma}}(e^{-(y-1)^2/(2\sigma^2)} + e^{-(y+1)^2/(2\sigma^2)}), \tag{6.35}$$

and

$$f(y) = \frac{P(y/1)}{P(y/0) + P(y/1)} = \frac{1}{(1 + e^{2y/\sigma^2})}. \tag{6.36}$$

The maximization in equation (6.33) can be done explicitly and the value of $\delta(y)$ at which the maximum occurs is given by

$$\delta^*(y) = \frac{4f(y)(1-f(y))\pi(y)}{1 + \sqrt{1 + 4Cf(y)(1-f(y))}}, \tag{6.37}$$

where C has to be chosen to satisfy the constraint $\int_0^\infty \delta(y)dy = \delta$.

6.3 Thresholds for the Ensemble of Random Codes

As an application of the above techniques, we present the typical set decoding thresholds for the ensemble of random binary codes. The ensemble of random binary linear codes with rate R has spectral shape

$$r(\delta) = H(\delta) + (R-1)\log 2. \tag{6.38}$$

We numerically obtain the typical set decoding thresholds over a range of R, and plot the threshold versus rate R, for the three choices of typical set decoder described above. Figure 6.3 displays the noise threshold as $E_s/N_0 = 1/(2\sigma^2)$, where E_s is the transmitted energy per channel symbol. Figure 6.4 displays the noise threshold expressed as $E_b/N_0 = 1/(2R\sigma^2)$, where E_b is the transmitted energy per information bit. (We have $E_s = RE_b$.) We see that of the three schemes, testing the mean and variance gives a better threshold (lower E_s/N_0 or E_b/N_0) than testing only the mean, and the third scheme of testing the distribution of samples is better than either of the other two, as expected.

A simplification occurs in solving this problem for random codes. We wish to find, for each R, the σ^2 such that

$$\sup_{0<\delta\leq 1} -K(\delta,\sigma^2) + H(\delta) + (R-1)\log 2 = 0. \tag{6.39}$$

Since R occurs as an additive parameter in $r(\delta)$, we can instead, for each σ^2, determine the rate R at which it is the threshold as

$$R = \inf_{0<\delta\leq 1} \frac{K(\delta,\sigma^2) - H(\delta)}{\log 2} + 1. \tag{6.40}$$

The function to be minimized here is convex and so the minimization is easily done numerically.

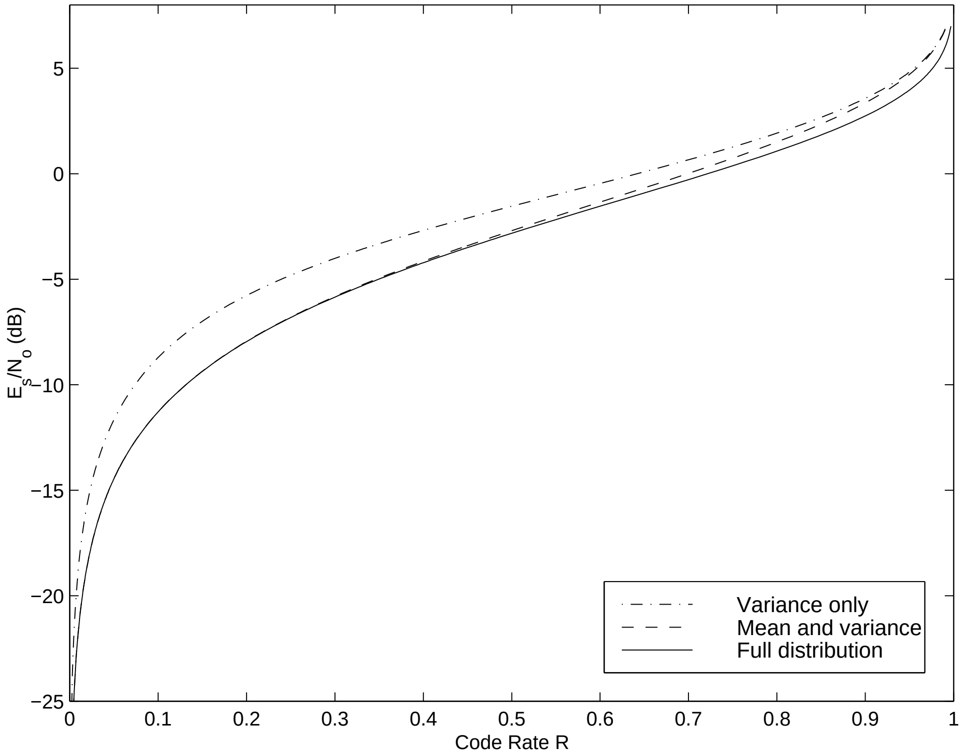

Figure 6.3: Noise thresholds, expressed as E_s/N_0, for random codes for the three choices of typical sets.

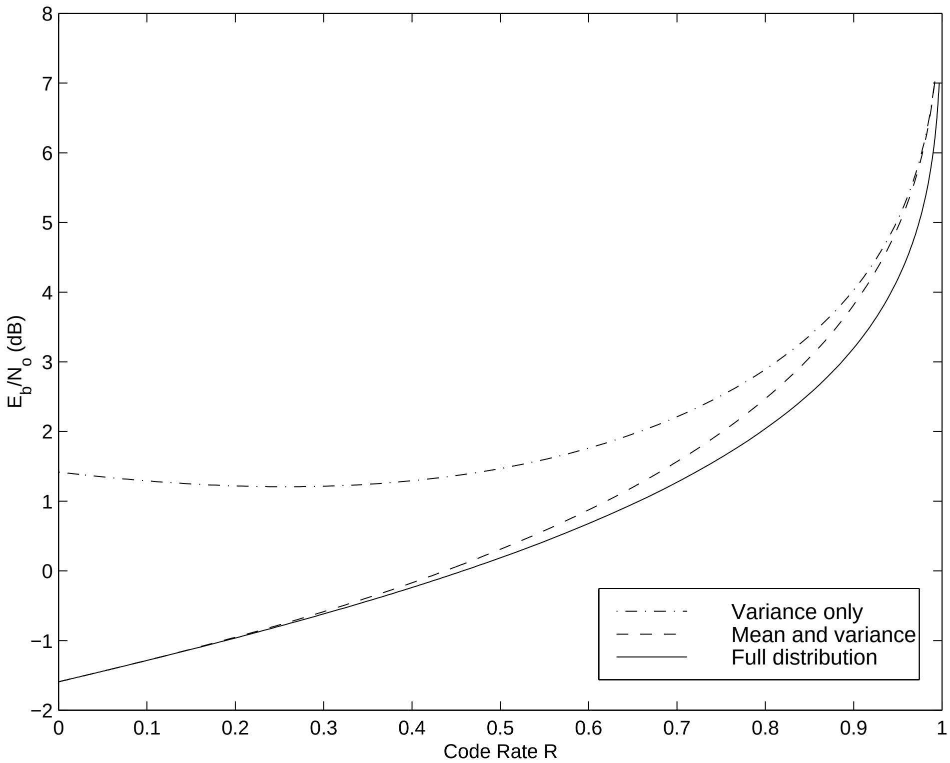

Figure 6.4: Noise thresholds, expressed as E_b/N_0, for random codes for the three choices of typical sets.

Bibliography

[1] S. Aji, G. Horn, and R. McEliece, "Iterative decoding on graphs with a single cycle," in *Proc. ISIT,* Boston, p. 276, MA, August 1998.

[2] S. M. Aji, G. B. Horn, R. J. McEliece, and M. Xu, "Iterative Min-Sum Decoding of Tail-Biting Codes," in *Proc. IEEE Information Theory Workshop,* Killarney, Ireland, pp. 68–69, June 1998.

[3] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory,* vol IT-46, pp. 325–346, March 2000.

[4] S. Aji, H. Jin, A. Khandekar, R. J. McEliece, and D. J. C. MacKay, "BSC thresholds for code ensembles based on 'Typical Pairs' Decoding," to appear in *Proc. IMA Workshop on Codes and Graphs*, August 1999.

[5] S. Aji, H. Jin, R. J. McEliece, and D .J. MacKay, "Typical set decoder on AWGN channel," submitted to *International Symposium on Information Theory and Its Applications,* Honolulu, Hawaii, November 2000.

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

[7] G. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding: Turbo codes," in *Proc. 1993 International Conf. Comm.,* Geneva, pp. 1064–1070, May 1993.

[8] D. Divsalar, "A Simple Tight Bound on Error Probability of Block Codes with Application to Turbo Codes," Jet Propulsion Laboratory TMO Progress Report 42-139, July-September 1999, pp. 1–35, November 15, 1999. (The TMO Progress Report is available at `http://tmo.jpl.nasa.gov/tmo/progress_report/` .)

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

[10] G. D. Forney, F. R. Kschischang, and B. Marcus, "Iterative decoding of tail-biting trellises." in *IEEE Information Theory Workshop,* San Diego, California, pp. 11–12, February 1998.

[11] B. J. Frey, "Bayesian networks for pattern classification, data compression, and channel coding," Ph.D. dissertation, Univ. Toronto, Toronto, ON, Canada, 1997.

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

[13] R. Gallager, *Low-Density Parity-Check Codes.* Cambridge, Mass: The M.I.T. Press, 1963.

[14] N. Jacobson, *Basic Algebra I.* New York: W. H. Freeman, 1985.

[15] F. V. Jensen, *An Introduction to Bayesian Networks.* New York: Springer-Verlag, 1996.

[16] F. V. Jensen and F. Jensen, "Optimal Junction Trees," in *Proc. of 10th Conf. on Uncertainty in Artificial Intelligence,* Seattle, Washington, pp. 360-366, July 1994.

[17] D. Lind and B. Marcus, *Symbolic Dynamics and Coding.* Cambridge University Press, 1996.

[18] S. L. Lauritzen and D. J. Spiegelhalter, "Local computation with probabilities on graphical structures and their application to expert systems," *J. Roy. Statist. Soc. B*, pp. 157–224, 1988.

[19] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *Cryptography and Coding, 5th IMA conference,* Springer Lecture notes in Computer Science no. 1025. Berlin: Springer-Verlag, 1995, pp. 100-111.

[20] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity-check codes," *Electronics Letters*, vol. 32, pp. 1645–1646, 1996. Reprinted in *Electronics Letters*, vol. 33, pp. 457–458, 1996.

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

[22] R. J. McEliece, R. B. Ash, and C. Ash, *Introduction to Discrete Mathematics*. New York: Random House, 1989.

[23] R. J. McEliece, D. J. C. MacKay, and J. -F. Cheng, "Turbo decoding as an instance of Pearl's 'belief propagation' algorithm," *IEEE J. Sel. Areas Comm.,* vol. 16, no. 2, pp. 140–152, February 1998.

[24] R. J. McEliece, E. R. Rodemich, and J-F. Cheng, "The turbo decision algorithm," in *Proc. 33rd Allerton Conference on Communication, Control and Computing,* pp. 366–379, October 1995.

[25] A. V. Oppenheim and R. W. Schafer, *Discrete-time signal processing*. Englewood Cliffs, N.J.: Prentice Hall, 1989.

[26] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*. San Mateo, CA: Morgan Kaufmann, 1988.

[27] G. R. Shafer and P. P. Shenoy, "Probability propagation," *Ann. of Math. and Art. Intel.*, vol. 2, pp. 327–352, 1990.

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

[29] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, April 1967.

[30] Y. Weiss, "Belief propagation and revision in networks with loops," MIT A.I. memo no. 1616, November 1997.

[31] Y. Weiss and W. T. Freeman, "Correctness of belief propagation in Gaussian graphical models of arbitrary topology," UC Berkeley CS Department TR UCB–CSD-99-1046, submitted to *Neural Computation*.

[32] W. T. Freeman and Y. Weiss, "On the fixed points of the max-product algorithm," MERL TR99-39, submitted to *IEEE Trans. Inform. Theory*.

[33] N. Wiberg, "Codes and Decoding on General Graphs," dissertation no. 440, Linköping Studies in Science and Technology, Linköping, Sweden, 1996.