SEQUENTIAL DECODING OF LINEAR BLOCK CODES

by

Dirk J. Tempel

A Thesis
Presented to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

March 1993

ISBN 0-315-81684-8

Name DIRK JAN TEMPEL.

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

ELECTRONICS AND ELECTRICAL ENGINEERING — SUBJECT TERM

0544 — SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Journalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0727
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0745
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0523
Psychology	0525
Reading	0535
Religious	0527
Sciences	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578
Ancient	0579
Medieval	0581
Modern	0582
Black	0328
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	0317
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760

EARTH SCIENCES

Biogeochemistry	0425
Geochemistry	0996
Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleoecology	0426
Paleontology	0418
Paleozoology	0985
Palynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575
Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Science	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463

Applied Sciences

Applied Mechanics	0346
Computer Science	0984
Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544 —
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994

PSYCHOLOGY

General	0621
Behavioral	0384
Clinical	0622
Developmental	0620
Experimental	0623
Industrial	0624
Personality	0625
Physiological	0989
Psychobiology	0349
Psychometrics	0632
Social	0451

SEQUENTIAL DECODING OF LINEAR BLOCK CODES

BY

DIRK J. TEMPEL

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Acknowledgments

I would like to thank Professor Ed Shwedyk for his help and support over the past two years. I would like to thank the University of Manitoba and its donors for the Duff Roblin Fellowship and the Clarence Bogardus Sharpe Memorial Scholarship. I would like to thank the Cable Telecommunications Research Institute for the Cable Telecommunications Research Fellowship. Finally, I would like to thank the Natural Sciences and Engineering Research Council of Canada for support.

Abstract

This thesis describes the use of the sequential stack algorithm for decoding linear block codes. The motivations for using this algorithm are that (i) it can be applied to any linear block code, (ii) it uses soft decisions, and (iii) it is efficient at moderate to high signal-to-noise ratios.

Because the sequential stack algorithm was developed to decode convolutional codes, it is not necessarily suited to block codes. Instead of modifying the algorithm to suit block codes, block code encoders can be designed to suit the algorithm. Two techniques for designing encoders are described in the thesis. The first follows from Wolf's trellis construction techniques, while the second casts a block code in the form of a rate one time-varying convolutional code. For suitable encoders, the sequential algorithm can be used to decode binary, q-ary, and concatenated codes. Further, it is proven that the sequential algorithm performs maximum likelihood soft decision decoding.

To be able to determine the effectiveness of the sequential algorithm, computational complexity measures are formulated. Using these measures, it is shown that the stack algorithm is the most efficient algorithm for decoding the (24,12) extended Golay code when the signal power is at least four times greater than the noise power. Computer simulations show that the stack algorithm can be made more efficient by setting a search limit and by reducing its stack size. These improvements come at the cost of error performance.

Overall, the thesis shows that the sequential algorithm is a viable alternative for decoding linear block codes at reasonable signal-to-noise ratios.

Table of Contents

List of Figures

List of Acronyms

AWGN	additive-white-Gaussian-noise
BPSK	binary phase shift keying
DMC	discrete memoryless channel
GF	Galois field
GIFO	greatest-in-first-out
LIFO	last-in-first-out
SNR	signal-to-noise ratio
SSA	sequential stack algorithm
WER	word error rate

Chapter 1 - Introduction

Owing to their algebraic properties, linear block codes are typically decoded using algebraic techniques. Generally, these algebraic techniques suffer from an inherent loss of 2 dB [Proa89] in error performance because they can not use soft decisions. This is not to say that all block codes can not be soft decision decoded, but when efficient algorithms exist they are usually specific to a code or class of codes at best. On the other hand, convolutional codes are easily decoded using the Viterbi algorithm or a sequential algorithm that use soft decisions and hence have a 2 dB advantage over block codes. Therefore, the ability to extend the convolutional decoding techniques to block codes would clearly be advantageous.

Why have soft decisions been so easy to use when decoding convolutional codes but not when decoding block codes? Perhaps the overwhelming algebraic structure of linear block codes has overshadowed the trellis structure that block codes also possess. It is the trellis structure of convolutional codes that allows soft decision decoding to be easily implemented. Thus, if a trellis structure can be assigned to a block code, then soft decision decoding will easily follow.

Wolf has shown how to construct a trellis for linear block codes in general [Wolf78]. The Viterbi algorithm may then be applied to this trellis to decode block codes using soft decisions. Unfortunately, the width of Wolf's trellis grows exponentially with the number of parity symbols used in the block code, thereby, making the Viterbi algorithm inefficient for higher parity codes. For these larger trellises a sub-optimum algorithm such as the M-algorithm [MaMo82] could be used. If the signal-to-noise ratios are at least moderate, then a sequential algorithm would be very efficient without sacrificing the optimal error performance of the Viterbi algorithm.

1.1 Sequential Algorithms

Since trellises for block codes are very wide, a sequential algorithm, working at moderate signal-to-noise ratios, is an effective decoding alternative to the Viterbi algorithm. Unlike the Viterbi algorithm, a sequential algorithm follows a single path through a trellis until it reaches the end of the trellis or it decides to abandon its current path and follow a better path. At reasonable signal-to-noise ratios, a sequential algorithm will only explore a small number of promising paths. As a result, the sequential algorithm will visit a fraction of the overall trellis that the Viterbi algorithm must cover and hence it is more efficient.

Akin to a tourist travelling through New York, the sequential algorithm requires a map, road signs, and a compass to successfully decode a block codeword. That is, the sequential algorithm requires a trellis for the block code (city map), information determining where the algorithm is in the trellis (road signs), and a measure telling the algorithm which direction it should follow (compass).

As important as maps, road signs, and compasses are, the tourist can not travel through New York unless he has transportation. Similarly, a block code can not be decoded unless there is a decoding algorithm. Because of its pedagogic simplicity the stack algorithm can be used to demonstrate the application of sequential decoding to block codes.

1.2 Review of Trellis Decoding Techniques

The idea of assigning a trellis to a block code first appeared in 1978 in papers by Wolf [Wolf78] and Massey [Mass78]. Both were motivated by the ability to use the Viterbi algorithm to decode linear block codes easily. While Wolf answered how to apply the Viterbi algorithm, Massey went a step further and showed how to measure the complexity of the Viterbi algorithm. Complexity could then be used as a guide for designing trellises for block codes.

Under Wolf's trellis construction, the width of the trellis grows exponentially with the number of parity symbols in the block code. As a result, the Viterbi algorithm is only computationally practical for low parity codes. Recognizing this, Matis and Modestino, in 1982, proposed using the M algorithm to search the trellis [MaMo82]. The M algorithm only searches a subset of the trellis and, therefore, has a lower complexity than the Viterbi algorithm. The disadvantage of the M algorithm (as compared to the Viterbi algorithm) is that it is not a maximum likelihood algorithm. Consequently, there is a trade-off between computational complexity and error performance.

The sequential algorithm is also adept at searching large trellises. In 1991, Offer and Perkins described the use of the sequential stack algorithm to decode systematic binary block codes [OfPe91]. In their algorithm, they use a modified Fano metric that makes the algorithm close to (but not exactly) maximum likelihood. The disadvantage of the sequential algorithm is that it has a variable complexity depending on the channel noise. This limits the algorithm to channels with reasonable noise levels.

All three approaches allow soft decision decoding to be performed for block codes. As well, the algorithms only require that a trellis (or a tree) structure can be assigned to the code, and, hence, are general soft decoding algorithms.

1.3 Other Soft Decision Decoding Algorithms

Other than the trellis decoding algorithms, there are only a few good soft decision algorithms that can be used to decode any linear block code. This overview will be limited to those algorithms that are designed to minimize the codeword error rate or at least do so asymptotically. These algorithms are: generalized Wagner decoding [SnBe89], generalized minimum distance decoding [Forn66], and channel measurement decoding [Chas72].

Given the received sequence, all of these algorithms perform the same three steps while decoding. First, hard decisions are made on the received sequence. In the second step, the algorithm determines a set of contending codewords. Finally, a codeword from this set is chosen using a soft decision criteria. The algorithms differ from each other in the manner in which the set of contending codewords is determined.

Generalized Wagner Decoding

In 1954, Silverman and Balser [SiBa54] considered the problem of choosing between a code (the Wagner code) that probably corrects all single errors and a code (the Hamming code) that definitely corrects all single errors. Though the latter initially appears to be the best choice, this decision is clouded by the fact that the probability that a bit is received in error is higher for the Hamming code. This observation is the consequence of an equal energy codeword assumption. The Wagner code is simply formed by appending a parity check bit to the information sequence. It is decoded by making hard decisions on the received sequence and then complementing the least likely bit if the parity fails. Because it only has one extra bit, as opposed to several, the probability of error for Wagner codes (using this decoding procedure) is less than that for Hamming codes (using a hard decision decoding algorithm).

In 1989, Snyders and Be'ery [SnBe89] showed how to generalize the Wagner decoding rule to make it useful for all codes (not just single parity codes). Though the algorithm can be used to decode linear block codes in general, it is most efficiently used for small parity codes. With this in mind, they have showed how to apply the algorithm to the cosets of a subcode and, thereby, create an efficient algorithm. The generalized algorithm can be summarized as follows: make hard decisions and evaluate the syndrome; secondly, among all of the sets of linearly independent columns of the parity check matrix that add to the syndrome, find the one for which the sum of confidence values is minimum; finally, complement the bits associated with the set found. This algorithm is the only one of the three algorithms considered in the overview that is optimum in regards to minimizing the word error rate.

Generalized Minimum Distance Decoding

In 1966, Forney [Forn66] presented a new distance criterion, called generalized minimum distance, that incorporates soft decision information. He then presented an algorithm that uses the generalized measure and is nearly maximum likelihood for a low noise white Gaussian channel. The algorithm works by erasing various numbers of the least reliable bits and then decoding the resultant words with an erasures-and-errors decoder. Of the resulting codewords, the one that falls within the generalized minimum distance of the received word is selected as the correct codeword.

Channel Measurement Decoding

In 1972, Chase [Chas72] presented three asymptotically optimum algorithms for soft decision decoding of block codes. These algorithms differ in the sizes of the sets of contending codewords they construct. In all algorithms, the sequence of hard decisions is perturbed by a set of test patterns. The perturbed sequences are then decoded using a hard decision algorithm. Of these candidates, the codeword that produces an error pattern of minimum analog weight is selected as the correct codeword.

1.4 Thesis Outline

Chapter 2 shows how to represent block codes as a tree or trellis. These structures can be constructed easily by using the block code's encoder. The sequential stack algorithm, as applied to binary, q-ary, and concatenated codes, is described in the remainder of the chapter.

The key motivation for using a sequential algorithm is that it is an efficient algorithm for optimal decoding of block codes at moderate signal-to-noise ratios. Thus, chapter 3 deals with the issue of computational complexity. Here, a previously defined measure of complexity is used to determine the complexity of the stack algorithm as applied to linear block codes.

Chapter 4 presents computer simulation results that show that the stack algorithm is optimal and efficient. The block codes used in the simulations are the (8,4) extended Hamming code and the (24,12) extended Golay code. For the (24,12) Golay code, the computational complexity is compared to the leading algebraic techniques. The simulations show that by 6 dB the sequential algorithm is the most efficient algorithm for soft decision decoding of the (24,12) Golay code.

Though the sequential algorithm is efficient, it can be made more efficient through refinements to the algorithm. These refinements usually come at the cost of error performance. Chapter 5 deals with two techniques to reduce the complexity of the sequential stack algorithm. These are setting an upper search limit and reducing the stack size.

Finally, chapter 6 contains conclusions and recommendations for further study.

Chapter 2 - Sequential Stack Algorithm for Block Codes

In section 1.2 the Viterbi, sequential, and M algorithms for decoding block codes were briefly discussed. Of these, the sequential algorithm is the only algorithm that can optimally (in terms of error performance) decode large parity block codes. The sequential algorithm is visualized as a tree (or possibly trellis) searching algorithm. Thus, in order to apply the algorithm to block codes, a tree representation must be found for block codes. In section 2.1 the tree and trellis representations for block codes are found. Tree representations can be found for any block code whether it is linear or non-linear. If the code is linear, then a trellis representation can be found. Finally, it is shown that the tree or trellis representation can be easily generated using the code's encoder. Section 2.2 shows how to design these encoders based on the code's generator matrix.

Sections 2.3 through 2.6 describe the sequential stack algorithm in detail for binary, q-ary, and concatenated codes. In section 2.4 the metric used while decoding is discussed. It is also shown that the sequential stack algorithm using this metric is optimal in terms of error performance.

2.1 Alternative Representations of Block Codes

Convolutional codes can be decoded using a tree or a trellis searching algorithm. These algorithms have the advantage of being general (applicable to all convolutional codes) and being able to incorporate soft decision information easily. Block codes can also be decoded using these algorithms if they are looked at in the proper manner. The tree or trellis representation is a trade of the use of the block code's algebraic structure for generality and soft decision decoding ease.

The usefulness of a tree or trellis representation is that it allows one to reduce the number of codewords under consideration. For example, for linear binary codes, a decision made at the fork between two branches halves the number of contending codewords. Rather than considering each codeword individually until a match is found, the codewords are reduced until only a single choice survives. This is akin to a game of twenty questions where each question narrows down the possibilities rather than a situation where all possibilities are considered individually until a match is found.

All block codes, linear or non-linear, have a tree representation as do all convolutional codes. This representation is the most general, and, consequently, ignores a large portion of the code's algebraic structure. Nevertheless, this representation does allow soft decision decoding to take place.

A tree for an (n,k) block code over GF(q) can be created conceptually rather easily. First, a tree representing all possible n-tuples with elements from GF(q) (q^n in total) can be grown. This tree will have the characteristic that each branch will have q exiting branches. This implies that the total number of branches at each level in the tree will be q times as large as the previous level. The last level of the tree will have a total of q^n branches. So far the only information from the code used to construct the tree has been the block size n and the field size q. The second step in constructing the proper tree for the (n,k) block code is to prune the tree so that all paths through the tree represent codewords from the (n,k) block code. The number of paths will be equal to the number of codewords which is usually equal to q^k.

As an example, the tree for a (4,2) block code over GF(2), where the code consists of the 4 codewords {0011, 0101, 1101, 1111}, can be constructed using the previous procedure. First a tree growing to 2^4 branches is built and labelled.

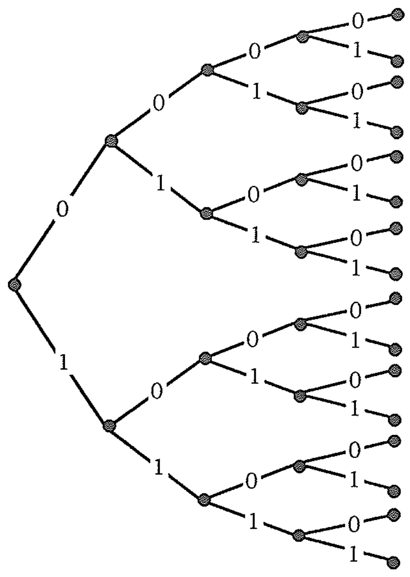


Figure 2.1.1. Full Binary Tree

Pruning all branches that do not belong to a specific codeword leaves the following tree:

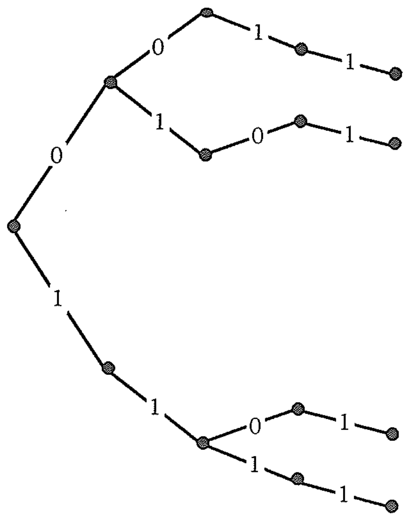


Figure 2.1.2. Pruned Binary Tree

Once a tree is constructed, any codeword can be found by tracing a path through the tree.

An even more compact representation of a block code can be constructed if the code is linear. A linear block code has a generator matrix G and parity matrix H. Using the parity matrix H, Wolf has shown how to construct a trellis. As an example, the trellis for a (5,3) block code is shown below. The zeros and ones that make up the code are represented by solid and dashed lines, respectively.

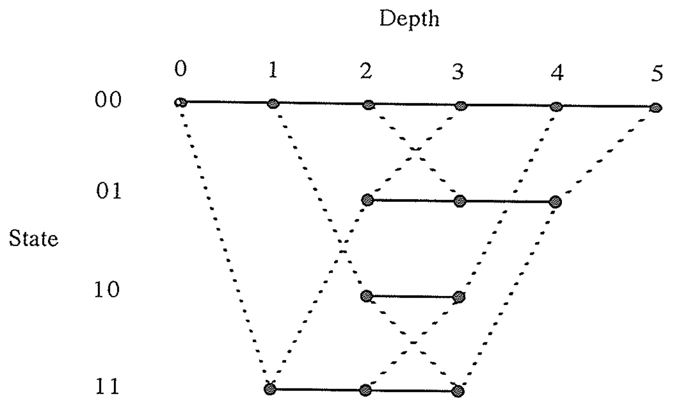


Figure 2.1.3. Block Code Trellis

2.2 Encoders

The following is quoted from the preface of Richard E. Blahut's coding textbook:

"Good codes need good decoders, and good decoding algorithms have been difficult to find. In the end, it may be just as fruitful for theoreticians to search out new codes to fit known decoders as it is to search out new decoders to fit known codes."[Blah83]

Therefore, instead of trying to suit the sequential decoding algorithm to block codes, it might be wiser and easier to try to suit the block codes to the decoder. That is, for an arbitrary block code, it might be better to find an equivalent block code that has an encoder suited to the decoding algorithm. With this goal in mind, it is necessary to first determine what the sequential algorithm requires of an encoder.

The object of the decoder is to retrace the path followed by the encoder through the tree or trellis. To accomplish this the decoder requires a replica of the original encoder to define the tree or trellis. Then the decoder can attempt to determine which information sequence was encoded based on the received noise corrupted codeword.

The first requirement of an encoder is that it can output a symbol for every input symbol. This is necessary for the algorithm to be able to generate the tree "on the fly". That is, the encoder does not have to wait for the entire information sequence before it can determine the beginning of a codeword.

It is important for the decoding algorithm to know where it is in the tree at all times. This information determines which branches exit the particular node at which the algorithm is currently stationed. The easiest method for this is to store the entire tree in a memory that can be referenced by the algorithm as it moves forward. For large codes this presents a practical problem since the tree grows exponentially with the number of information symbols.

When the Viterbi algorithm is used on a trellis for a block or convolutional code it keeps track of where it is by storing the state of the encoder. The successor states can then be determined for all of the encoder's possible inputs. Thus, the second requirement for an encoder is that it has a state that can be read.

In summary, though the details of the encoder are not important, it is necessary that the encoder be able to output a symbol for every input symbol, and also have a state that can be recognized and read by the decoder. As long as a "black box" encoder satisfying these two requirements can be found, the algorithm can keep track of where it is at all times.

Though the "black box" encoder is sufficient to guarantee that the algorithm can move through the tree without storing the tree it may not be the simplest encoder. Further, for an arbitrary encoder it may be difficult to determine the state of the encoder. Two different approaches for designing encoders follow.

Approach 1

This approach follows from Wolf's trellis construction techniques [Wolf78]. For (n,k) linear block codes, at least, a one-input-one-output encoder can be designed using the code's generator or parity matrix. First, since every linear block code has an equivalent systematic code, it is only necessary to consider systematic block codes. Then the first k output symbols are equal to the first k input symbols. The remaining (n-k) output symbols are equal to the elements of the vector formed by the weighted sum of the first k columns of the code's systematic parity check matrix. The sum is weighted by the first k input symbols. This weighted sum is also the state of the encoder. Mathematically, after t symbols have entered the encoder, the state is:

$$\mathbf{s}_t = \sum_{j=0}^{t-1} i_j \mathbf{h}_j = i_0\mathbf{h}_0 + i_1\mathbf{h}_1 + \cdots + i_{t-1}\mathbf{h}_{t-1} \tag{2.2.1}$$

where $\mathbf{h}_j$ is the jth column of the code's parity check matrix. Figure 2.2.1. shows a diagram of this type of encoder.

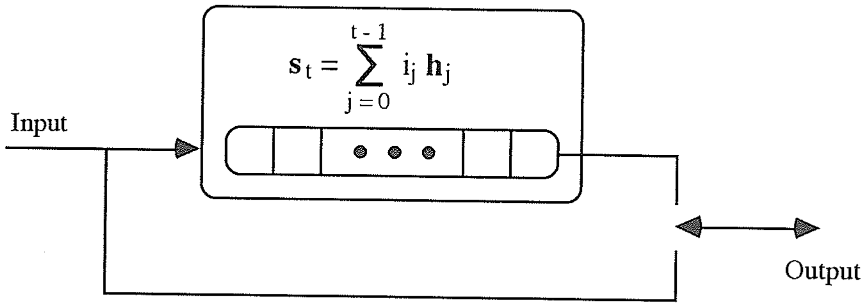


Figure 2.2.1. Encoder for Systematic Block Codes

If the code is cyclic, then it can also be encoded systematically with the following shift register circuit with feedback where the g_i's are determined by its generator polynomial.

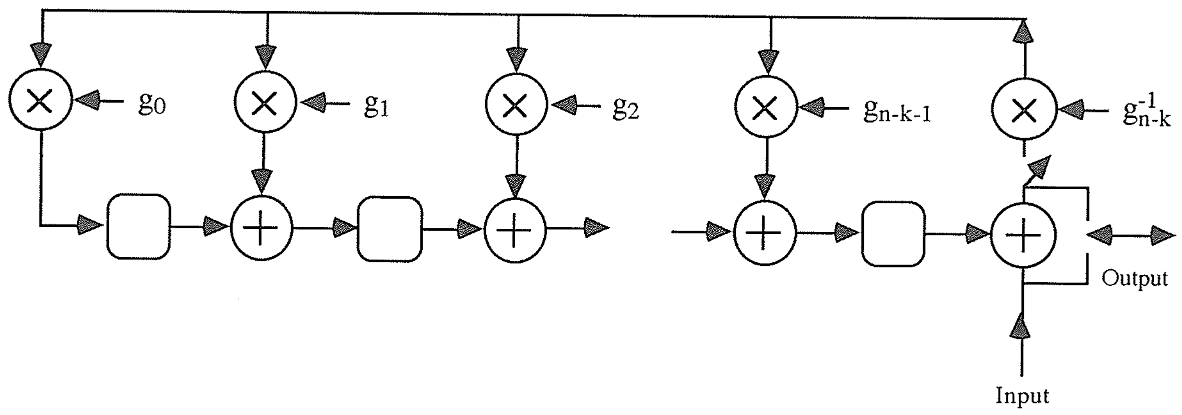


Figure 2.2.2. Systematic Encoder for Cyclic Codes

Initially the output is connected to the input and the feedback loop is closed. After k symbols have been entered into the encoder, the output is connected to the encoder circuit and the feedback loop is opened. The state of this encoder is defined by the contents of the (n-k) shift registers.

As a simple example, the encoder for an (n,n-1) Wagner code [SiBa54] is shown below. The feedback loop is switched out after (n-1) symbols have been entered into the encoder. This circuit simply appends a single even parity bit to any binary sequence.

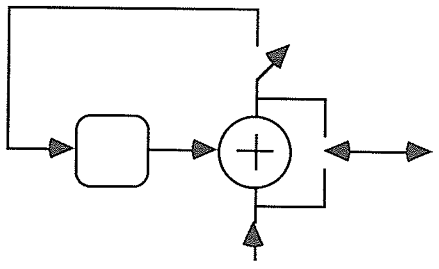
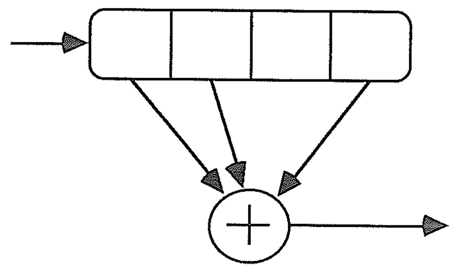


Figure 2.2.3. Encoder for a Wagner Code

Approach 2

A second approach to designing encoders for block codes is motivated by the fact that the sequential algorithm was originally designed for convolutional codes. If a block code can be made to look like a convolutional code, then it can be decoded in a more straight forward manner.

For example, encoders for a cyclic block code and a convolutional code are shown in Figure 2.2.4. Aside from the upper taps of the convolutional code, both encoders are identical. Both codes have the same trellis and tree representations differing only in that the one for the block code has a single bit labelling each branch instead of two.


(7,4) Block Code or (1,1,3) Convolutional Code

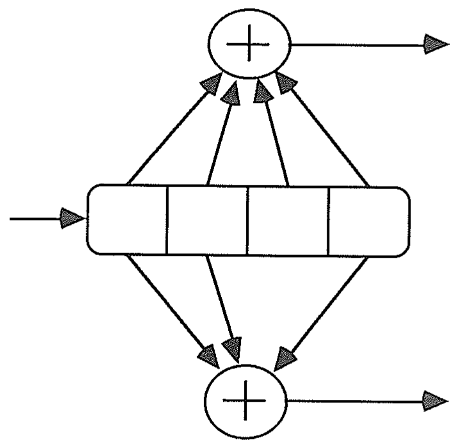

(2,1,3) Convolutional Code

Figure 2.2.4. Block and Convolutional Code Encoders

Motivated by the above example, if it is possible to encode all linear block codes (and not only cyclic codes) using a sequence of shift registers, then the encoding and decoding of block codes will be the same as for convolutional codes. The following theorem shows how to find an equivalent code that can be encoded using a sequence of shift registers. This theorem is proved in Appendix 2-A.

Theorem 1

Given an (n,k) linear block code with generator matrix G, an equivalent code can be found that has a generator matrix of the following form:

$$G = \begin{bmatrix} \leftarrow \mathbf{g}_0 \rightarrow 0\,0\,0 \cdots 0\,0 \\ 0 \leftarrow \mathbf{g}_1 \rightarrow 0\,0 \cdots 0\,0 \\ \vdots \qquad \vdots \qquad \vdots \\ 0 \cdots 0\,0 \leftarrow \mathbf{g}_{k-2} \rightarrow 0 \\ 0 \cdots 0\,0\,0 \leftarrow \mathbf{g}_{k-1} \rightarrow \end{bmatrix}$$

where the $\mathbf{g}_i$'s are (n-k+1)-tuples.

This equivalent code can be encoded using (n-k) shift registers with time-varying taps, as shown in Figure 2.2.5. The taps are determined by the columns of the generator matrix. That is, the taps for the first information symbol are determined by the first column of the generator matrix while the taps for the second symbol are determined by the second column. For example, the taps for the kth symbol are the first element of $\mathbf{g}_{k-1}$, the second element of $\mathbf{g}_{k-2}$, the third element of $\mathbf{g}_{k-3}$, and so on until all (n-k+1) taps are determined. More precisely, $g_i(t) = g_{t-i,t}$ for $0 \le t-i \le k-1$, and is equal to zero otherwise. The state of the encoder is simply the contents of the (n-k) shift registers.

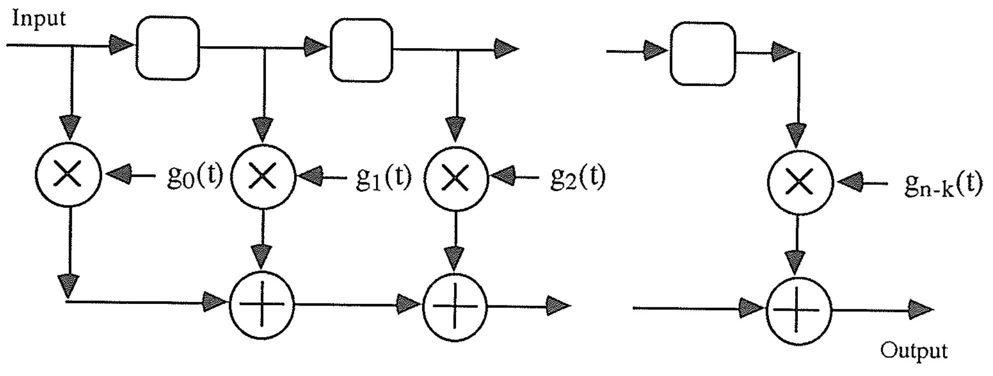


Figure 2.2.5. Encoder for Block Codes

The procedure for using this encoder is as follows:

1. Initialize all the registers to zero.
2. Set the encoder taps.
3. Enter a symbol.
4. Repeat steps 2 and 3 for k information symbols.
5. Repeat steps 2 and 3 for (n-k) zeros.

This encoder yields n codeword symbols due to the k information symbols and the (n-k) zeros. The block encoder has only one output for every input. Therefore, the block code can be viewed as a rate one (i.e. n=1 and k=1) time-varying convolutional code and can be decoded as if it were a convolutional code.

Cyclic codes are a special case of the previous theorem. If the code is cyclic, then all of the (n-k+1)-tuples in G are the same, and, hence, the encoder taps do not change with time. The encoder is shown in Figure 2.2.6.

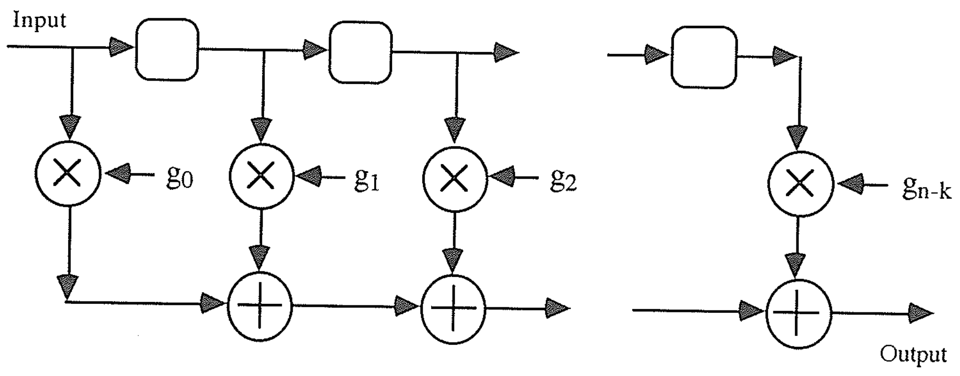


Figure 2.2.6. Encoder for Cyclic Codes

Both approaches have their own advantages and disadvantages. The first approach has the advantage that the code it generates is systematic so it is easy to remove the information from the code. A second advantage is that it is easy to label the tree or trellis for the code for depths less than k because the first k output symbols are equal to the first k input symbols. The disadvantage of this first approach is that the trellis described by the encoder is irregular. The second approach has the advantage that its trellis is the familiar trellis for a corresponding convolutional code. It only differs in the branch labels. As well, since the analogy to convolutional codes is so much stronger, it is clearer how to use this encoder with the sequential algorithm.

2.3 Sequential Stack Algorithm (SSA) for Binary Block Codes

The object of the sequential algorithm is to search through the tree constructed for the code to find the codeword that most closely resembles the received sequence. The sequential stack algorithm follows a path through the tree until some other path looks more promising. At this point the other path is explored until a better, if any, path is found. This continues until some path reaches the end of the tree at which time this path is chosen as the path most resembling the noise corrupted codeword. During the search, partially searched paths are stored in a stack and arranged according to their resemblance to the received sequence.

Instead of storing the partial paths on a stack that is sorted, the paths can be stored in a priority queue as described by Chang and Yao [ChYa86]. The priority queue is a greatest-in-first-out (GIFO) type of stack as opposed to the usual last-in-first-out (LIFO) stack. The advantage of the priority queue is that it avoids a lot of the needless sorting required by the stack algorithm by realizing that the most promising path is the only path required by the algorithm at any given time. As well, Chang and Yao's parallel implementation of the priority queue allows the algorithm to determine the most promising partial path in a fixed processing time.

The algorithm for decoding an (n,k) binary block code is:

1. Delete the best path from the priority queue.
2. If the path is at the end of the tree, output the information sequence and quit. Otherwise:
3. Initialize the state of the encoder.
4. Input a bit (0 or 1) into the encoder.
5. Calculate the branch metric using the encoder output bit and the received bit.
6. Add the branch metric to the path metric.
7. Append the input bit to the information sequence.
8. Increment the path length.
9. Store the metric, path, path length, and state in the priority queue.
10. Repeat steps 3 thru 9 until all of the possible input bits are exhausted.
11. Return to step 1.

The algorithm presented above requires some explanation. Before any decoding can take place a priority queue of sufficient size must be available to the algorithm. The priority queue must be able to store at least four pieces of information about a particular node. These are the metric, partial path, path length, and encoder state. The metric is the measure used to determine which path most resembles the received sequence. The partial path contains the information bits leading to the current state of the encoder, and the path length is the number of information bits. The priority queue must then be initialized with the root of the code tree where the metric will be zero, the partial path will be empty, the path length will be zero, and the state will be the all-zero state. In this way step 1 will delete the root of the code tree and begin the decoding process from this point.

Step 1 requires that the best path be deleted from the priority queue. The four pieces of information that come with that best path must then be stored for use in later steps. Specifically, in step 3 the state of the encoder is initialized. Here the encoder's state is initialized to the state of the path deleted from the priority queue in step 1. In step 6 the path metric from step 1 is needed to calculate a new path metric. In steps 7 and 8, the partial path and path length from step 1 are updated.

Block codes are encoded by serially entering k information bits into the encoder followed by (n-k) zeros. Therefore, as step 10 suggests, there are two possible inputs bits (i.e. 0 or 1) for path lengths less than k but only one (i.e. 0) for paths with longer lengths.

2.4 Fano Metric - Block Codes Over GF(2)

In order to decode a received sequence, a measure is needed to determine which of the possible codewords the received sequence most resembles. When sequentially decoding convolutional codes the usual measure is the Fano metric. This metric reflects the different depths of paths through the trellis. Before the sequential algorithm can be applied to a block code, the Fano metric must be formulated. The metric for decoding binary block codes transmitted over a memoryless channel is presented below and derived in Appendix 2-B.

Fano Metric - Binary Codes

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} log\left[1 + \frac{p(r_j \mid \overline{c_{ij}})}{p(r_j \mid c_{ij})} \right] \qquad (2.4.1)$$

where r_j is the jth output of the receiver's matched filter, c_{ij} is the jth bit in the ith codeword, c_{ij} with an overbar is the complement of c_{ij}, N_i is the length (in bits) of ith partial codeword, and $p(r_j \mid c_{ij})$ is the channel's conditional output probability density function.

As a specific example, the Fano metric for decoding binary block codes transmitted over an AWGN channel using BPSK modulation is:

Fano Metric - AWGN/BPSK

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} log\left[1 + exp\left(-\frac{4\,(2c_{ij} - 1)\,r_j\sqrt{E_s}}{N_o}\right)\right] \qquad (2.4.2)$$

where E_s is the energy per transmitted bit, and N_o is the single sided noise power density. It should also be noted that since the metrics can be scaled by any constant the logarithm can be with respect to any base.

Theorem 2

The sequential stack algorithm with the Fano metric always finds the maximum likelihood codeword.

Proof

The goal of the decoder is to find the codeword with maximum probability given the received sequence. Mathematically, given the received sequence **r**, the decoder attempts to find the codeword $\mathbf{c}_i$ that maximizes $\Pr(\mathbf{c}_i \mid \mathbf{r})$ or $\log \Pr(\mathbf{c}_i \mid \mathbf{r})$, assuming that all codewords are equally likely to be transmitted. If the Fano metric is calculated for all codewords, then the codeword with the maximum Fano metric will also be the codeword that maximizes $\Pr(\mathbf{c}_i \mid \mathbf{r})$.

The stack algorithm does not calculate the total metric for all paths, though. Therefore, it is necessary to show that if a completed path reaches the top of the stack (because it has the largest metric) then no other path can have a metric that is larger than its metric. In other words, if a completed path reaches the top of the stack then it must be the most likely codeword. This can be proved by making the observation that the branch metric is always less than or equal to zero for block codes. Consequently, if a completed path is compared to a partial path, where the path metric of the former is larger than the metric of the latter, then it is impossible for the path metric of the partial path to ever exceed the metric of the completed path. Since all other paths through the tree must diverge from the completed path, the completed path will be the maximum likelihood codeword.

The proof leads to the following corollary.

Corollary

The sequential stack algorithm with any branch metric, that is less than or equal to zero, will always find the maximum likelihood codeword, provided that maximizing the metric is equivalent to maximizing $Pr(\mathbf{c}_i \mid \mathbf{r})$.

For example, the following metrics may also be used for decoding codewords transmitted over an AWGN channel with BPSK modulation:

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} \left[\sqrt{E_s} - (2c_{ij} - 1)\, r_j \right]^2 \qquad (2.4.3)$$

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} \left[|r_j| - (2c_{ij} - 1)\, r_j \right] \qquad (2.4.4)$$

The Fano metric was developed with the goal of minimizing the amount of searching performed by the sequential algorithm. Despite the fact that these two metrics guarantee optimal error performance they are not guaranteed to minimize the amount of searching. A comparison of these and other possible metrics for sequential decoding can be found in Appendix 2-C.

2.5 SSA for Block Codes over GF(q)

Block codes over larger fields than GF(2) can also be decoded using the previous algorithm. Unlike the approach taken by Offer and Perkins [OfPe91], there is no need to change the code to a binary code with this algorithm. Consequently, any block code can be decoded using the algorithm and not just codes over fields that are a power of 2.

A few logical changes have to made to the algorithm to generalize it from GF(2) to GF(q). First of all, the tree for a block code over GF(q) has q exiting branches per node rather than two. As a result q different inputs must be entered into the encoder and q paths must be added to the stack per step.

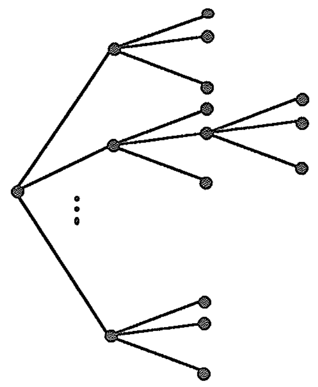


Figure 2.5.1. q-ary Tree

The algorithm for decoding block codes over GF(q) is then:

1. Delete the best path from the priority queue.
2. If the path is at the end of the tree, output the information sequence and quit. Otherwise:
3. Initialize the state of the encoder.

4. Input a **symbol** from **GF(q)** into the encoder.
5. Calculate the branch metric using the encoder output and received **symbol**.
6. Add the branch metric to the path metric.
7. Append the input **symbol** to the information sequence.
8. Increment the path length.
9. Store the metric, path, path length, and state in the priority queue.
10. Repeat steps 3 thru 9 until all of the possible input **symbols** are exhausted.
11. Return to step 1.

This algorithm differs from that for binary block codes in that it is generalized to symbols over GF(q) instead of bits from GF(2). For example, in step 10 there are q possible input symbols, as opposed to two possible input bits, for path lengths less than k (the number of information symbols). For paths of k symbols or more there is still only one possible input symbol, namely zero. The differences are noted by the bold type.

The metric used for decoding must change to reflect the fact that the code is over GF(q) instead of GF(2). The Fano metric now becomes:

Fano Metric - q-ary Codes

$$M(i,\mathbf{r}) = \sum_{j=1}^{N_i} \log \frac{p(r_j \mid c_{ij})}{\sum_{c_{ij}} p(r_j \mid c_{ij})} \qquad (2.5.1)$$

where c_{ij} is a symbol from GF(q). The derivation of the metric is shown in Appendix 2-B.

2.6 SSA for Concatenated Codes

Concatenated codes are generally decoded using two decoders that decode the inner and outer codes. When a convolutional code is used as the inner code, the Viterbi algorithm or the sequential algorithm can be used to decode the code using soft decisions. Usually, the outer code is a block code, like a Reed-Solomon code, that is used to correct the burst errors of the Viterbi or sequential decoder. The decoder for the Reed-Solomon code does not use soft decisions at all, though. Since it is now possible to decode a block code using soft decisions via the Viterbi or sequential algorithm it may be possible to decode the overall concatenated code with these algorithms. The motivation for using a single algorithm to decode the entire code is twofold. The first motivation is the ability to easily use soft decisions in the decoding. Secondly, this approach results in the savings of one decoder.

A block diagram of a block-convolutional concatenated coding scheme is shown below. The outer code is an (N,K) block code with symbols from GF($q=p^m$) while the inner code is an (n,k) convolutional code with symbols from GF(p).

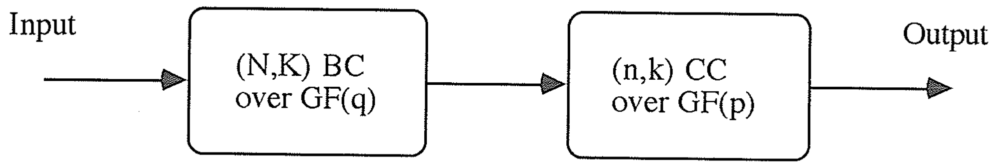


Figure 2.6.1. Concatenated Coding Scheme

For every symbol from GF($q=p^m$) that is an input into the outer (block) encoder, m symbols from GF(p) are output and passed to the inner (convolutional) encoder. These m symbols then cause nm/k symbols from GF(p) to be output from the convolutional encoder. Therefore, one symbol from GF($q=p^m$) input into the overall encoder causes mn/k symbols from GF(p) to be output. At this point it is necessary to impose the constraint that k divides m so that an integral number of symbols are output at a time.

This overall encoder can then be used to construct a tree for the concatenated code. This tree will look the same as the tree for the outer (block) code. It will only differ in that the branches will be labelled with the nm/k symbols from GF(p) instead of a single symbol from GF($q=p^m$). As well, the convolutional code introduces memory between successive codewords. The implication of this is that the trees of successive codewords will be joined one after the other. An example of a tree for a concatenated block-convolutional code is shown below.

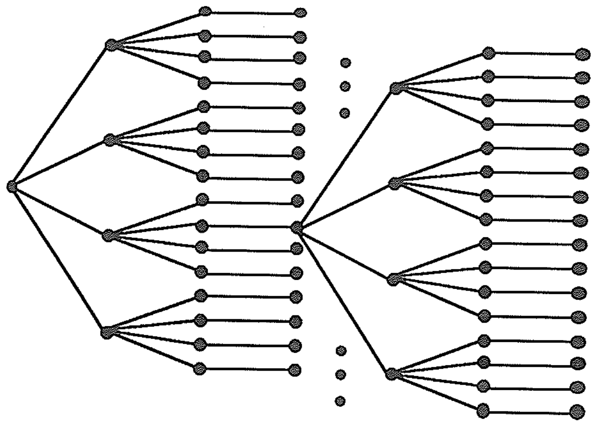


Figure 2.6.2. Concatenated Code Tree

The state of the overall encoder is the state of the block encoder and the state of the convolutional encoder. The state of the block encoder will be over GF($q=p^m$) while the state for the convolutional code will be over GF(p). Given an (N,K) block code with q^{N-K} states and an (n,k) memory M convolutional code with p^{kM} states, then the overall encoder will have $p^{m(N-K)+kM}$ states. This enormous number of states effectively eliminates the Viterbi algorithm as a decoding alternative leaving only the sequential algorithm.

The algorithm for decoding concatenated block-convolutional codes is:

1. Delete the best path from the priority queue.
2. If the path is at the end of the tree, output the information sequence and quit.

Otherwise:

3. Initialize the state of the encoder.
4. Input a symbol from GF($q=p^m$) into the encoder.
5. Calculate the branch metric using the encoder output and received **symbols**.
6. Add the branch metric to the path metric.
7. Append the input symbol to the information sequence.
8. Increment the path length.
9. Store the metric, path, path length, and state in the priority queue.
10. Repeat steps 3 thru 9 until all of the possible input symbols are exhausted.
11. Return to step 1.

There are three key differences between this algorithm and that for block codes over GF(q). First, in step 5 the metric is calculated using several symbols over GF(p) instead of a single symbol over GF(q). Second, there are q possible input symbols for path lengths between 0 and K, N and N+K, 2N and 2N+K etc. where N and K are the dimensions of the outer block code. Finally, the metric for decoding the concatenated code is the Fano metric that would be used for decoding the inner (n,k) convolutional code over GF(p).

Though convolutional codes are not restricted to the binary field most of the popular convolutional codes are binary. If binary convolutional codes are used, then the outer block code must be over a field that is a power of 2. As well, the algorithm implementation requires that k divides m. The easiest way to guarantee this is to choose k to be equal to one. This constraint is usually met since most convolutional codes used in concatenated schemes are rate 1/n codes. Then, for every symbol from GF($q=2^m$) input into the overall encoder, nm symbols from GF(2), or simply nm bits, are output from the overall encoder.

2.7 Summary

Algorithms that use soft decisions while decoding block codes are difficult to find. For convolutional codes, at least, two algorithms exist: these are the Viterbi algorithm and the sequential algorithm. The Viterbi algorithm is applied to the code's trellis and the sequential algorithm is normally applied to the code's tree. All block codes have a tree representation, and if the code is linear, then it will also have a trellis representation. Therefore, using the Viterbi or sequential algorithm is a viable decoding alternative for block codes.

Instead of modifying the algorithms to suit block codes, equivalent codes can be found that suit the algorithms. As the algorithm moves through the tree or trellis it must keep track of where it is and where it can proceed. This can be done by storing the tree or trellis. A better way to do this is to require that the code's encoder produce one output symbol for every input symbol and have a state that can be read and initialized. Two encoders satisfying these criteria are presented in section 2.2. The first is a systematic encoder while the second is simply a rate one time-varying convolutional code encoder.

Once the encoders are designed, the algorithms can be applied in their usual forms. Descriptions of the sequential algorithm as applied to binary, q-ary, and concatenated codes are provided in sections 2.3 through 2.6. The advantages of this approach, as compared to Offer and Perkins [OfPe91], are that it allows block codes over any field to be decoded and that it has a stronger resemblance to the algorithm for convolutional codes.

Finally, the sequential algorithm requires a metric to determine the most promising path to extend: this metric is the Fano metric. For block codes, the Fano metric allows the sequential algorithm to perform maximum likelihood decoding.

Appendix 2-A Generator Matrix for Block Codes

Theorem

Given an (n,k) linear block code with generator matrix G, an equivalent code can be found that has a generator matrix of the following form:

$$G = \begin{bmatrix} \leftarrow \mathbf{g}_0 \rightarrow 0\,0\,0 \cdots 0\,0 \\ 0 \leftarrow \mathbf{g}_1 \rightarrow 0\,0 \cdots 0\,0 \\ \vdots \qquad \vdots \qquad \vdots \\ 0 \cdots 0\,0 \leftarrow \mathbf{g}_{k-2} \rightarrow 0 \\ 0 \cdots 0\,0\,0 \leftarrow \mathbf{g}_{k-1} \rightarrow \end{bmatrix}$$

where the $\mathbf{g}_i$'s are (n-k+1)-tuples.

Given the generator matrix of an (n,k) linear block code, it can always be put in the form G = [I | P], where I is a k x k identity matrix and P is a k x (n-k) matrix. Define $G_i = [\, I \mid P_i \,]$ as the matrix formed by removing the first i rows and columns of G = [I | P]. If (k-i)>(n-k), then the first (n-k+1) rows of P_i must be linearly dependent, and some combination of these will add to zero. If this addition is carried out, then the (n-k+1)-tuple, $\mathbf{g}_i$, will be formed.

When (k-i)≤(n-k) the matrix formed by the last (k-i-1) columns of P_i must have linearly dependent rows. These should be added to produce an all-zero (k-i-1)-tuple. The addition will form the (n-k+1)-tuple, $\mathbf{g}_i$. The procedure fails if an all-zero row is encountered. This can be avoided by interchanging columns of P before the algorithm is applied so that:

$p_{j+2k-n,\, j} = 1 \qquad \text{for } 0 \le j \le n-k-1$

if there are no 1's to the right of this position (note: p_{ij} is the element found in the ith row and jth column of P).

Appendix 2-B Fano Metric for Block Codes

The Fano metric, used when sequentially decoding convolutional codes, can be expressed as [ViOm79]:

$$M(i,\mathbf{r}) = \sum_{j=1}^{N_i} \left[\log_2 \frac{p(r_j \mid c_{ij})}{p(r_j)} + \frac{1}{N_i} \log_2 \pi_i \right] \tag{2.B.1}$$

where r_j is jth matched filter output, c_{ij} is the jth symbol in the ith codeword, $p(r_j)$ is the channel output probability density function, N_i is the number of symbols that have been partially decoded and π_i is the probability of transmitting those previous N_i symbols.

The channel output probability density function can be found by averaging the conditional output density function $p(r_j \mid c_{ij})$ over all channel inputs. That is,

$$p(r_j) = \sum_{c_{ij}} p(c_{ij})\, p(r_j \mid c_{ij}) \tag{2.B.2}$$

For equally likely channel inputs from a q-ary input alphabet:

$$p(c_{ij}) = \frac{1}{q} \quad (2.B.3) \qquad \text{and} \qquad \pi_i = q^{-N_i} \quad (2.B.4)$$

Therefore,

$$p(r_j) = \frac{1}{q} \sum_{c_{ij}} p(r_j \mid c_{ij}) \tag{2.B.5}$$

Substituting for π_i and $p(r_j)$ in the Fano metric expression gives:

$$M(i,\mathbf{r}) = \sum_{j=1}^{N_i} \log_2 \frac{p(r_j \mid c_{ij})}{\sum_{c_{ij}} p(r_j \mid c_{ij})} \tag{2.B.6}$$

which is the appropriate version of the Fano metric for decoding block codes over GF(q).

For an alphabet where q is a power of a prime (i.e. $q = p^m$) each of the q symbols can be represented by a stream of m symbols from GF(p). If the codewords are now transmitted using their GF(p) representation, then the codewords become m times as long. The previous Fano metric can still be used to decode the codewords now represented over GF(p) simply by letting N_i be the number of symbols over GF(p), as opposed to GF(q), that have been partially decoded. Typically q is a power of 2 so each symbol over GF(q) can be represented by m bits. Thus, N_i is the number of bits that have been partially decoded. For this case the Fano metric becomes:

$$M(i,\mathbf{r}) = \sum_{j=1}^{N_i} \log_2 \frac{p(r_j \mid c_{ij})}{p(r_j \mid c_{ij}) + p(r_j \mid \overline{c_{ij}})} \qquad (2.B.7)$$

where c_{ij} with the over bar is the complement of c_{ij}.

This last metric can be derived using an alternate approach. The Fano metric for decoding binary convolutional codes is [ViOm79]:

$$M(i,\mathbf{r}) = \sum_{j=1}^{N_i} \left[\log_2 \frac{p(r_j \mid c_{ij})}{p(r_j)} - R \right] \qquad (2.B.8)$$

where R is the rate of the code. The notion that a block code is a rate one convolutional code suggests that R = 1 should be substituted in the above expression. Substituting for R and $p(r_j)$ will then give the metric that was derived above (see equation 2.B.7). Defining p_{ij} as:

$$p_{ij} = \frac{p(r_j \mid \overline{c_{ij}})}{p(r_j \mid c_{ij})} \qquad (2.B.9)$$

simplifies the metric to:

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} \log_2 [\, 1 + p_{ij} \,] \qquad (2.B.10)$$

which is the Fano metric for decoding 2^m-ary block codes.

For signals transmitted over an additive-white-Gaussian-noise (AWGN) channel using binary phase shift keying (BPSK), the conditional channel output density function is:

$$p(r_j \mid c_{ij}) = \frac{1}{\sqrt{\pi N_o}} \exp\left[- \frac{(r_j - (2c_{ij} - 1) \sqrt{E_s})^2}{N_o} \right] \qquad (2.B.11)$$

where E_s is the energy per transmitted bit, N_o is the single sided noise power density, and c_{ij} is either 0 or 1. Therefore, p_{ij} is:

$$p_{ij} = \exp\left(- \frac{4\,(2c_{ij} - 1)\, r_j \sqrt{E_s}}{N_o} \right) \qquad (2.B.12)$$

and the Fano metric for decoding 2^m-ary block codes transmitted over an AWGN channel using BPSK modulation is:

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} \log_2 \left[1 + \exp\left(- \frac{4\,(2c_{ij} - 1)\, r_j \sqrt{E_s}}{N_o} \right) \right] \qquad (2.B.13)$$

Appendix 2-C Other Metrics

The following metrics are derived for binary codewords transmitted over an AWGN channel using BPSK modulation.

Metric 1

The maximum likelihood codeword can be found by finding the codeword for which the correlation between the transmitted sequence and the received sequence is the largest. When using BPSK modulation this is given by:

$$M(i,\mathbf{r}) = \sum_{j=1}^{n} (2c_{ij} - 1)\, r_j \qquad (2.C.1)$$

More generally, the maximum likelihood codeword is the codeword that maximizes the following expression where A and B are constants and B is strictly positive.

$$M(i,\mathbf{r}) = A + B \sum_{j=1}^{n} (2c_{ij} - 1)\, r_j \qquad (2.C.2)$$

Setting A and B equal to:

$$A = -\sum_{j=1}^{n} |r_j| \qquad (2.C.3) \qquad \text{and} \qquad B = 1 \qquad (2.C.4)$$

gives the following metric:

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} \left[\, |r_j| - (2c_{ij} - 1)\, r_j \,\right] \qquad (2.C.5)$$

In comparison to the Fano metric, this new metric has an interesting interpretation. This metric is simply a scaled asymptotic approximation to the Fano metric. For an

AWGN channel with BPSK modulation, the Fano metric for decoding binary block codes is:

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} \log\left[1 + \exp\left(-\frac{4\,(2c_{ij} - 1)\, r_j \sqrt{E_s}}{N_o}\right)\right] \tag{2.C.6}$$

As the argument of the exponential tends to negative infinity the exponential tends to zero, and, hence, the metric tends to zero. As the argument tends to positive infinity the metric becomes linear in r_j. Approximating the metric by these limits leads to the following metric.

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} \left[\,|r_j| - (2c_{ij} - 1)\, r_j\,\right] \tag{2.C.7}$$

This is the metric that was derived above. As a result, this metric should be expected to perform (in terms of complexity) very closely to the Fano metric.

Metric 2

Alternatively, a more intuitive choice (as compared to the above choices) for the constants A and B can be made. The constants A and B can be selected as:

$$A = -\sum_{j=1}^{n} \sqrt{E_s} \quad (2.C.8) \qquad \text{and} \qquad B = 1 \quad (2.C.9)$$

This gives the following metric:

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} \left[\,\sqrt{E_s} - (2c_{ij} - 1)\, r_j\,\right] \tag{2.C.10}$$

Here, the term $\sqrt{E_s}$ can be interpreted as a bias that causes the algorithm to search more paths through the tree. Unfortunately, this metric is not solely negative so it will not always find the maximum likelihood codeword. Simulations have shown that using this metric costs about 0.1 to 0.3 dB in error performance.

Changing the bias can improve error performance slightly but also degrade error performance significantly. Increasing the bias improves performance while decreasing the bias degrades performance. This can be explained by realizing that a larger bias will cause the algorithm to follow a path less deeply into the tree before switching to a new path. As a result more paths will be examined and hence the chance of finding the correct path improves. An alternate explanation is that the larger bias increases the likelihood that the metric will be negative. If the branch metrics are all negative then the algorithm will find the maximum likelihood codeword. If a branch metric is positive then the maximum likelihood codeword may or may not be found.

Metric 3

The maximum likelihood codeword is also the codeword that has the minimum squared euclidean distance to the received sequence. This is equivalent to finding the codeword with the maximum negative squared distance. Thus, the following metric will also yield the maximum likelihood codeword when used with the sequential stack algorithm.

$$M(i,\mathbf{r}) = -\sum_{j=1}^{N_i} \left[\sqrt{E_s} - (2c_{ij} - 1)\, r_j \right]^2 \qquad (2.C.11)$$

Comparisons

By Theorem 2, in section 2.4, metrics 1 and 3 must be maximum likelihood. It was already noted that metric 2 is close to maximum likelihood but is not. Metrics 1 and 3 can be compared to the Fano metric in terms of complexity. The following graph shows the average nodes searched, when decoding the (24,12) Golay code, as a function of the signal-to-noise ratio.

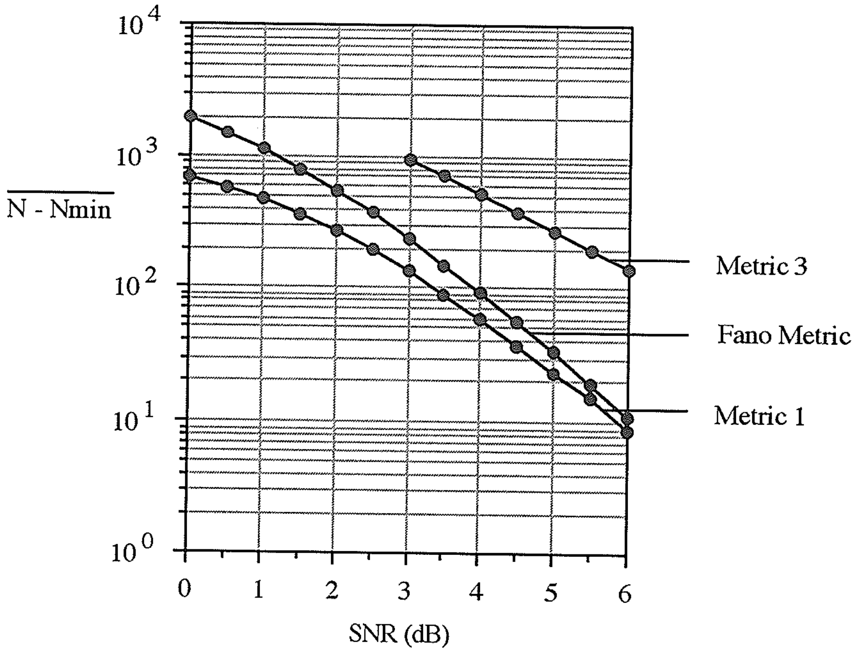


Figure 2.C.1. Average Nodes Searched for 3 Metrics

Metric 1 outperforms both metric 3 and the Fano metric in terms of average nodes searched. Since metric 1 is so closely related to the Fano metric, its complexity is only slightly better than that for the Fano metric.

Chapter 3 - Computational Complexity

To determine the effectiveness of the sequential stack algorithm for decoding block codes an objective measure must be formulated. Such a measure is the computational complexity of the algorithm. If the computational complexity can be measured, then it can be used as a basis for comparison between the sequential algorithm and other decoding alternatives.

Unlike the Viterbi algorithm, or most other block decoding techniques, the sequential algorithm performs a variable amount of work that depends on the noise level. That is, the sequential algorithm proceeds rather quickly and efficiently if the noise is low, but as the noise increases the sequential algorithm becomes inefficient. When used with convolutional codes, the complexity is normally measured in terms of the average number of branches or nodes searched by the algorithm for a given signal-to-noise ratio. This can also be measured when the algorithm is applied to block codes. In order to make a meaningful comparison to other block decoding techniques, though, this measure has to be translated into a measure for block codes.

3.1 A Complexity Measure for Block Codes

A reasonable definition for the computational complexity of a decoding algorithm is the total number of equivalent real number additions it performs. This obviously includes real number additions but also includes real number comparisons where the former and latter are given equal value. The definition does not account for binary operations, hard decisions, absolute values, or bit confidence calculations.

This definition has been used by Snyders and Be'ery to measure the complexity of their block decoding algorithms [SnBe89]. Consequently, any comparison of the sequential algorithm to those algorithms should be made using this definition.

The sequential algorithm manipulates four pieces of information (viz. state, metric, path, and path length). Of these, only the metric is a real number, and, thus, metric operations solely comprise the complexity.

3.2 Complexity Measured

There are only two metric operations in the stack algorithm. The first is metric addition which is performed when the branch metric is added to the path metric. The second operation is metric comparison which is performed in the priority queue after every deletion or insertion. The overall complexity is the sum of these two different operations. If the number of metric additions is denoted as N_m and the number of metric comparisons as N_c, then the complexity is $N_m + N_c$.

3.3 Metric Addition

For every node in the tree that the algorithm visits a new metric must be calculated. Thus, the number of metric additions is equal to the number of nodes visited. Denoting the number of nodes visited by N implies that:

Number of Metric Additions

$$N_m = N \qquad (3.3.1)$$

Minimum Number of Additions

Depending on the noise, the sequential algorithm will visit a number of nodes that falls between a minimum and a maximum number. The minimum number of nodes visited occurs when the algorithm does not backtrack. Up to a depth k into the tree, each node has two exiting branches (for (n,k) binary block codes) implying that two further nodes must be visited. For greater depths, a single branch leaves each node. Thus, the minimum number of nodes visited, and, hence, metric additions, is:

$$N_m^{min} = 2k + (n - k) \tag{3.3.2}$$

For block codes over GF(q) this generalizes to:

$$N_m^{min} = qk + (n - k) \tag{3.3.3}$$

Maximum Number of Additions

The maximum number of nodes visited occurs when the algorithm completely searches every path through the tree. The total number of nodes visited can be calculated by counting all the nodes in the code's tree. An example of a binary code's tree representation is shown in Figure 3.3.1.

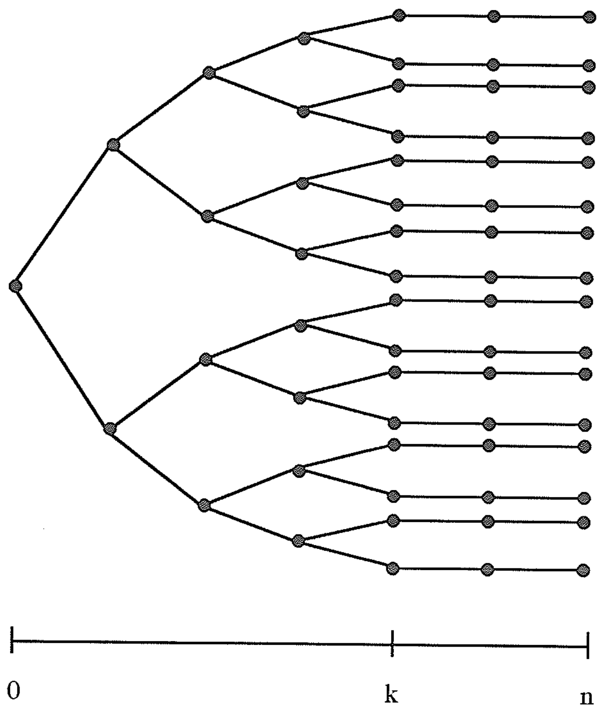


Figure 3.3.1. Tree for an (n,k) Binary Block Code

For the first k steps the tree doubles in size with each step. At depth k the tree will have 2^k terminal branches representing the 2^k possible codewords. For the (n-k) parity bits the terminal branches are simply extended by one branch per step which maintains a total of 2^k terminal branches at each step. The maximum number of nodes visited can then be calculated by counting all the nodes in the tree. Therefore, the maximum number of metric additions is:

$$N_m^{max} = 2 + 4 + \cdots + 2^k + (n - k)\, 2^k = (n - k + 2)\, 2^k - 2 \qquad (3.3.4)$$

For q-ary block codes this becomes:

$$N_m^{max} = q + q^2 + \cdots + q^k + (n - k)\, q^k = \left(n - k + \frac{q}{q - 1} \right) q^k - \frac{q}{q - 1} \qquad (3.3.5)$$

3.4 Metric Comparison

Metric comparisons occur whenever data is inserted or deleted from the priority queue. Visiting a node in the tree involves deleting and extending a node from the queue. Thus, the number of comparisons is related to the number of nodes visited. Before the comparisons can be counted, though, it is necessary to explain briefly the operation of the priority queue. The priority queue described by Chang and Yao [ChYa86] is guaranteed to output the maximum number that is stored in the queue. This is accomplished by comparing pairs of adjacent elements after every insertion or deletion operation. The two elements in the pair are reordered if the element furthest from the top of the queue is greater. Thus, if the queue contains N elements, N/2 (assuming N is even) comparisons need to be made.

Given the number of nodes visited, the number of comparisons can be upper bounded by considering a stack that initially increases in size and then remains constant in size. The stack increases in size if two nodes are entered between deletions. This corresponds to nodes in the tree with two exiting branches (i.e. depths less than k). Given that N nodes are visited assume that the stack grows for the first αN ($0 \leq \alpha \leq 1$) nodes. Since two nodes must be entered for the stack to grow by one, the stack grows to $\alpha N/2$ entries and the average stack size is approximately $\alpha N/4$. Deletion and insertion operations in the priority queue require that adjacent pairs of entries in the stack be compared. Hence, for a stack with N elements, N/2 comparisons must be performed. Therefore, the total number of comparisons is:

$N_c^{(1)}$ = (# deletions and insertions/step)(average # comparisons/del or ins)(# steps)

$$N_c^{(1)} \approx 3\left(\frac{\frac{\alpha N}{4}}{2}\right)\left(\frac{\alpha N}{2}\right) = \frac{3}{16}\alpha^2 N^2 \qquad (3.4.1)$$

The stack does not grow if only one node is inserted after a deletion. This corresponds to nodes in the tree with only one exiting branch (i.e. depths of k or greater). Knowing that the stack grew for the first αN nodes implies that it remains fixed at $\alpha N/2$ entries for the remaining $(1-\alpha)N$ nodes. Therefore, the total number of comparisons is:

$N_c^{(2)}$ = (# deletions and insertions/step)(average # comparisons/del or ins)(# steps)

$$N_c^{(2)} = 2\left(\frac{\frac{\alpha N}{2}}{2}\right)\left((1-\alpha)N \right) = \frac{1}{2}\alpha(1-\alpha)\, N^2 \qquad (3.4.2)$$

Combining, gives the total number of comparisons as:

$$N_c = N_c^{(1)} + N_c^{(2)} \approx \frac{1}{2}\left[\alpha - \frac{5}{8}\alpha^2\right]N^2 \qquad (3.4.3)$$

Finally, maximizing the expression over α bounds the number of comparisons.

Number of Metric Comparisons - Binary Codes

$$N_c \leq \frac{1}{5} N^2 \tag{3.4.4}$$

For block codes over GF(q), every deletion is followed by either 1 or q insertions. If αN nodes are visited at depths less than k, then $\alpha N/q$ steps are taken where the stack grows by (q-1) entries per step. As a result, the stack grows to $(q-1)\alpha N/q$ entries and averages about $(q-1)\alpha N/2q$ entries. For depths greater than or equal to k it remains fixed at $(q-1)\alpha N/q$ entries . Therefore, the total number of comparisons is:

$$N_c^{(1)} \approx (q+1)\left(\frac{\frac{(q-1)\alpha N}{2q}}{2}\right)\left(\frac{\alpha N}{q}\right) = \frac{q-1}{q}\frac{q+1}{4q}\alpha^2 N^2 \tag{3.4.5}$$

and

$$N_c^{(2)} = 2\left(\frac{\frac{(q-1)\alpha N}{q}}{2}\right)\left((1-\alpha)N\right) = \frac{q-1}{q}\alpha(1-\alpha)N^2 \tag{3.4.6}$$

Combining gives:

$$N_c = N_c^{(1)} + N_c^{(2)} \approx \frac{q-1}{q}\left[\alpha - \frac{3q-1}{4q}\alpha^2\right]N^2 \tag{3.4.7}$$

Maximizing over α gives:

Number of Metric Comparisons - q-ary Codes

$$N_c \leq \frac{q-1}{3q-1} N^2 \tag{3.4.8}$$

The assumption that the stack grows continually before staying at a fixed size is not necessarily how the algorithm operates in practice. This assumption implies that the algorithm is restricted to depths of k or greater once it begins working in this region. Instead, the algorithm is free to move anywhere in the tree. However, this reality does not contradict the bound on the number of comparisons. Even though the nodes are mixed they can be grouped into nodes at depths less than k and those at depths greater or equal to k. Since αN nodes at depths less than k are still visited, the stack growth remains the same and $N_c^{(1)}$ does not change. The difference is that the stack is not fully grown when nodes at depths of k or greater are visited. Therefore, this part of the complexity is less than $N_c^{(2)}$, and the actual complexity is less than the bound which, in turn, reaffirms the bound.

The bound on the number of comparisons is a quadratic bound. That is, it grows as the square of the number of nodes searched. This quadratic growth in complexity is a definite disadvantage of the stack algorithm and sequential algorithms in general. Other algorithms such as the Viterbi algorithm or M algorithm have a linear growth in complexity [AnMo91]. This is not to say that the sequential algorithm should not be used. In fact, as long as the noise level is low, the number of nodes visited (N) is small and consequently the number of comparisons is small as well. Therefore, the sequential algorithm is best used at moderate to high signal-to-noise ratios.

Minimum Number of Comparisons

Under the most favorable noise conditions the sequential algorithm will still have to perform a minimum amount of searching. A formula for the minimum number of nodes searched is given in section 3.3. The minimum number of comparisons in the priority queue, occurring when a minimum number of nodes are visited, can be calculated by considering the situation that led to the minimum number of nodes being visited. The number of comparisons at depths less than or equal to k can be calculated using equation

3.4.5 where αN is equal to qk. Substituting gives:

$$N_c^{(1)} \approx (q - 1) \frac{q + 1}{4} k^2 \qquad (3.4.9)$$

The number of comparisons at depths greater than k can be calculated with equation 3.4.6 where $(1-\alpha)N$ is equal to (n-k). This gives:

$$N_c^{(2)} = (q - 1)(n - k)\, k \qquad (3.4.10)$$

The minimum number of comparisons is then:

$$N_c^{min} = N_c^{(1)} + N_c^{(2)} \approx (q - 1) \left[(n - k)\, k + \frac{k^2}{2} \right] + \frac{(q - 1)^2}{4} k^2 \qquad (3.4.11)$$

Maximum Number of Comparisons

Under the worst noise conditions the algorithm may search the entire code tree. If this occurs, the algorithm searches a number of nodes that can be calculated using equation 3.3.5. The maximum number of comparisons occurs when the algorithm searches every node up to a depth k and then searches every node at greater depths. The first part of this can be calculated using equation 3.4.5 where αN is the number of nodes at depths less than or equal to k. This is equal to:

$$\alpha N = q + q^2 + \cdots + q^k = \frac{q}{q - 1} q^k - \frac{q}{q - 1} \approx \frac{q}{q - 1} q^k \qquad (3.4.12)$$

Substituting gives:

$$N_c^{(1)} \approx \frac{q + 1}{4(q - 1)} q^{2k} \qquad (3.4.13)$$

The second part can be calculated using equation 3.4.6 where $(1-\alpha)N$ is equal to the number of nodes at depths greater than k. That is,

$$(1 - \alpha) N = (n - k)\, q^k \qquad (3.4.14)$$

This gives:

$$N_c^{(2)} = (n - k)\, q^{2k} \tag{3.4.15}$$

The maximum number of comparisons is then given by:

$$N_c^{max} = N_c^{(1)} + N_c^{(2)} \approx \left(n - k + \frac{q + 1}{4(q - 1)} \right) q^{2k} \tag{3.4.16}$$

3.5 Hardware (Stack) Complexity

The hardware complexity is determined by the size of the stack required for the algorithm. Since the algorithm only stores the head of a path through the tree, the stack must be able to hold the heads of all possible paths through the tree. This corresponds to being able to hold each of the code's possible codewords. Therefore, for (n,k) block codes over GF(q), the priority queue or stack must be able to hold q^k entries for optimal decoding. For large codes, this stack size quickly becomes an obstacle to the optimal implementation of the SSA for block codes. Reducing the stack size (at the cost of error performance) is investigated in chapter 5.

3.6 Summary of Complexity Measures

This section summarizes the previous computational complexity results. Specifically, the minimum, maximum, and average complexities for decoding block codes over GF(q) are presented.

Based on the above discussion, the minimum complexity to decode an (n,k) block code over GF(q) consists of N_m metric additions and N_c metric comparisons that can be calculated using the following formulas.

Minimum Complexity

$$N_m^{min} = qk + (n - k) \tag{3.6.1}$$

$$N_c^{min} \approx (q - 1)\left[(n - k)\, k + \frac{k^2}{2} \right] + \frac{(q - 1)^2}{4}\, k^2 \tag{3.6.2}$$

Though the minimum complexity looks especially good, the maximum complexity is especially poor. Fortunately, for reasonable signal-to-noise ratios, the complexity of the stack algorithm does not approach its maximum. The approximate maximum number of metric additions (nodes searched) and comparisons are given below.

Maximum Complexity

$$N_m^{max} \approx (n - k)\, q^k \tag{3.6.3}$$

$$N_c^{max} \approx (n - k)\, q^{2k} \tag{3.6.4}$$

If the stack algorithm visits N nodes, then the number of metric additions is equal to N and the number of comparisons is upper bounded by cN^2, where c is constant determined by the stack sorting algorithm. For a priority queue, c is equal to (q - 1)/(3q - 1). For an algorithm that finds the maximum metric in the stack differently, c is not necessarily the same. For example, for a maximum finder (see Appendix 3-A), c equals (q - 1)/(4q - 2).

Mathematically, the overall complexity of the stack algorithm, N_{SA}, is bounded by:

Complexity

$$N_{SA} \leq N + cN^2 \qquad (3.6.5)$$

and the average complexity of the stack algorithm is bounded by:

Average Complexity

$$\overline{N_{SA}} \leq \overline{N} + c\overline{N^2} \qquad (3.6.6)$$

$$\overline{N_{SA}} \leq \sum_{N = N_{min}}^{N_{max}} \left(N + cN^2 \right) Pr[\, N \,] \qquad (3.6.7)$$

where N_{min} and N_{max} are:

$$N_{min} = qk + (n - k) \qquad (3.6.8)$$

$$N_{max} = \left(n - k + \frac{q}{q - 1} \right) q^k - \frac{q}{q - 1} \qquad (3.6.9)$$

and Pr[N] is the probability of visiting N nodes at a given signal-to-noise ratio.

Appendix 3-A The Maximum Finder

The priority queue is a significant improvement over a sorted stack largely because it exploits the fact that the stack algorithm only needs the maximum value in the stack at a given time. The priority queue also has the advantage of being parallel. If the priority queue can not be implemented in its parallel fashion then it loses this advantage. If this is the case, then the priority queue, though much better than a sorted stack, is not the most efficient algorithm for finding the maximum number in a list. Instead, a maximum finder which goes through the list sequentially is more efficient with the stack algorithm. The reason for this is that the maximum finder only needs to be used when a path needs to be deleted from the stack as opposed to the priority queue which performs comparisons after every deletion and insertion operation.

For a given stack size N a maximum finder requires (N - 1) comparisons to delete the best path. This is done by taking the first element in the stack and comparing it to the second. Then the largest of these is compared to the third. Of these, the largest is compared to the fourth. This continues until the last element in the stack is compared to the largest up to that point.

Unlike the maximum finder the priority queue rearranges the stack after every insertion and deletion. There are two patterns of insertions and deletions that occur with the sequential stack algorithm. One pattern is an insertion followed by a deletion. Assuming the stack size is N after the insertion implies that the number of comparisons needed is N/2 if N is even or (N - 1)/2 if N is odd. After the deletion the stack size will be (N - 1) and the number of comparisons is (N - 2)/2 if N is even or (N - 1)/2 if N is odd. For either even or odd N the total number of comparisons for an insertion-deletion pattern is (N - 1). This is equal to the number of comparisons performed by the maximum finder. Thus, for this pattern, the priority queue or maximum finder are equally effective in terms of the total number of comparisons. The second pattern is q (for q-ary codes) insertions followed by a single deletion. Since each insertion causes the priority queue to compare adjacent elements in the stack, q insertions will require more comparisons than a single insertion. Thus, for these patterns, the priority queue will require more operations than the maximum finder. For example, a pattern of 2 insertions followed by a single deletion requires that $3(N - 1)/2$ comparisons be performed. This is 50% greater than the number of comparisons performed by the maximum finder. Since the decoding will involve a mix of the two types of insertion-deletion patterns the priority queue will average somewhere between 0 and 50% more comparisons than a maximum finder.

For (n,k) block codes over GF(q) the number of comparisons as a function of the number of nodes searched can be bounded for the maximum finder. If the algorithm visits N nodes assume that αN of those are at depths less than or equal to k. At these depths, q nodes will be inserted into the stack for every node deleted. The following table lists the stack size and number of comparisons performed as a function of the number of nodes visited.

Nodes	Stack size	Comparisons
q	q	$q-1$
$2q$	$2q-1$	$2(q-1)$
$3q$	$3q-2$	$3(q-1)$
.	.	.
.	.	.
$(\alpha N/q)q$		$(\alpha N/q)(q-1)$

The number of comparisons is equal to:

$$N_c^{(1)} = (q - 1)\left[1 + 2 + \cdots + \frac{\alpha N}{q} \right] \approx \frac{q - 1}{q}\frac{2}{4q}\alpha^2 N^2 \qquad (3.A.1)$$

For the remaining $(1-\alpha)N$ nodes, at depths greater than k, the number of comparisons will still be given by equation 3.4.6. Adding gives the total number of comparisons.

$$N_c = N_c^{(1)} + N_c^{(2)} \approx \frac{q-1}{q}\left[\alpha - \frac{4q-2}{4q}\alpha^2\right]N^2 \qquad (3.A.2)$$

If this is maximized over α, then the number of comparisons is bounded as:

Number of Metric Comparisons - Maximum Finder

$$N_c \leq \frac{q-1}{4q-2}N^2 \qquad (3.A.3)$$

In comparison to the result for the priority queue, the maximum finder still requires a number of comparisons that grows as the square of the number of nodes searched. The difference is that the proportionality constant is less for the maximum finder.

The minimum number of comparisons with the maximum finder occurs when the sequential algorithm does not backtrack. For depths less than or equal to k, the total number of comparisons can be calculated using equation 3.A.1 where αN is equal to qk. Substituting for αN gives:

$$N_c^{(1)} \approx (q-1)\frac{2}{4}k^2 \qquad (3.A.4)$$

The total number of comparisons at greater depths for a maximum finder is the same as for a priority queue (see equation 3.4.10). Adding gives the total number of comparisons.

$$N_c^{min} \approx (q-1)\left[(n-k)\,k + \frac{k^2}{2}\right] \qquad (3.A.5)$$

As compared to a priority queue (see equation 3.4.11), the minimum number of comparisons with the maximum finder is less by an amount equal to:

$$N_c^{PQ} - N_c^{MF} = \frac{(q-1)^2}{4}k^2 \qquad (3.A.6)$$

As with the priority queue, the maximum number of comparisons occurs when the algorithm searches every node up to a depth k and then searches every node at greater depths. For the maximum finder, the first part is calculated using equation 3.A.1. where αN is approximated by equation 3.4.12. Substituting gives:

$$N_c^{(1)} \approx \frac{2}{4(q-1)} q^{2k} \tag{3.A.7}$$

The second component of the number of comparisons remains the same (see equation 3.4.15). Consequently, the maximum number of comparisons is:

$$N_c^{max} \approx \left(n - k + \frac{2}{4(q-1)} \right) q^{2k} \tag{3.A.8}$$

As expected, the number of metric comparisons still grows as q^{2k}.

Chapter 4 - Simulation Results

The two block codes selected for simulation were the (8,4) extended Hamming code and the (24,12) extended Golay code. For both codes, the simulations were performed assuming an AWGN channel with BPSK modulation. The discrete time equivalent of this channel and modulation scheme is shown in Figure 4.0.1. The all-zero codeword was transmitted a total of 10^5 times over this channel in order to insure a reliable estimate of the block code's error rate. For error rates of 10^{-4}, the error rate estimate has a 95% confidence of being within 20% of the code's true error rate; for larger error rates, the estimate improves [HeNo88]. The noise was generated using a pseudo-random Gaussian number generator with a period of two billion [PFTV88].

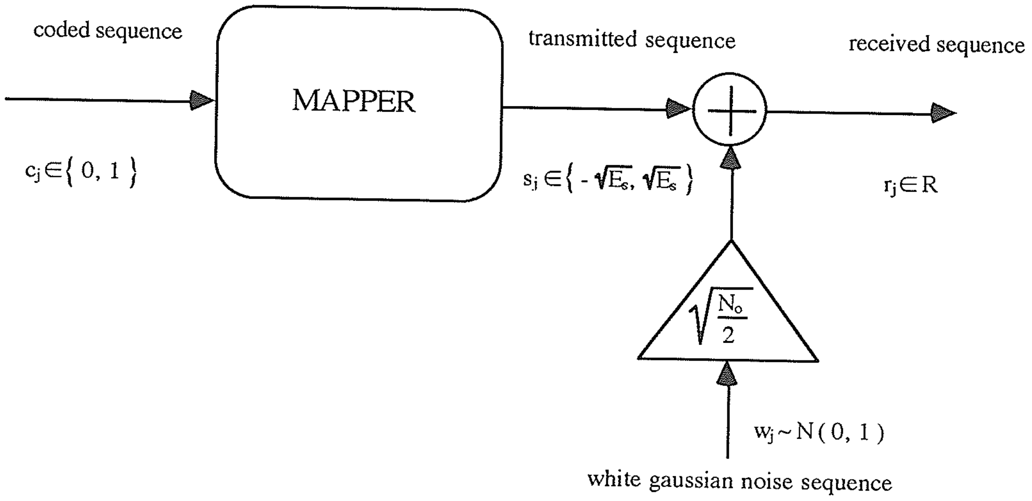


Figure 4.0.1. Discrete Time Channel

4.1 Encoders

The (8,4) Hamming code and the (24,12) Golay code are extended cyclic codes.

Their encoders are formed by cascading the encoders of their respective cyclic codes with Wagner code encoders. Appending an (8,7) Wagner encoder to the (7,4) Hamming encoder gives the overall encoder for the (8,4) extended Hamming code as shown in Figure 4.1.1. The state is defined by the contents of four shift registers, and, hence, the trellis travelled through by the stack algorithm will contain 2^4 states.

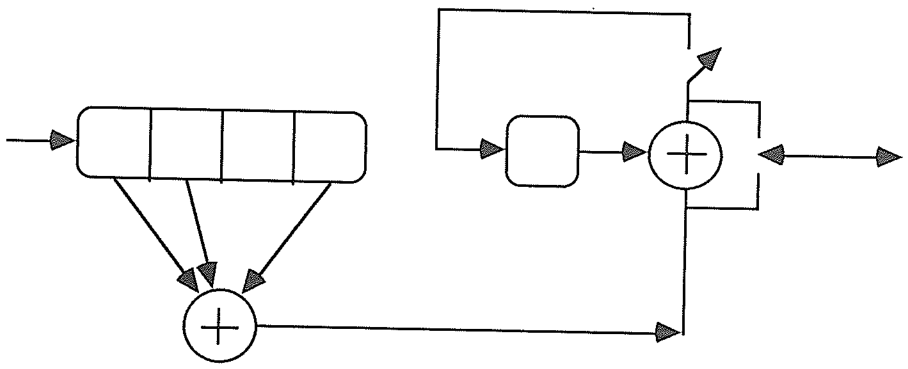


Figure 4.1.1. (8,4) Extended Hamming Code Encoder

The (8,4) code does not show the true value of a sequential algorithm since the Viterbi algorithm can easily handle a trellis with 16 states. To show the advantages of a sequential algorithm, a code with a larger trellis must be considered. The (24,12) extended Golay code has a trellis with 2^{12} states. The encoder for the (24,12) extended Golay code is shown in Figure 4.1.2.

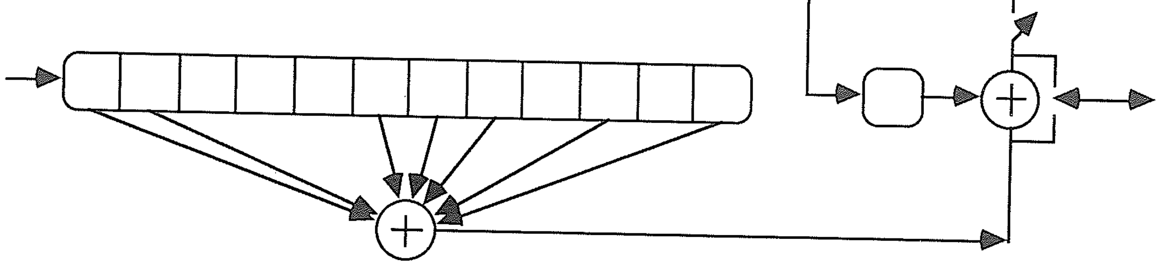


Figure 4.1.2. (24,12) Extended Golay Code Encoder

4.2 Error Performance

When the (8,4) extended Hamming code is transmitted over an AWGN channel using BPSK modulation, the codewords can be considered as a set of M = 16 equally likely biorthogonal signals. Therefore, the following result from Van Trees [VaTr69], for a set of M equally likely biorthogonal signals with energy E, can be used to calculate the error performance of a maximum likelihood soft decision decoder.

$$\Pr(\varepsilon) = 1 - \int_{0}^{\infty} \frac{1}{\sqrt{\pi N_o}} \exp\left[-\frac{1}{N_o}(x - \sqrt{E})^2\right]\left[\int_{-x}^{x} \frac{1}{\sqrt{\pi N_o}} \exp\left(-\frac{y^2}{N_o}\right) dy\right]^{\frac{M}{2}-1} dx \qquad (4.2.1)$$

Since 8 bits, each with energy E_s, are transmitted per codeword, the total energy of a codeword is $8E_s$. Substituting $E = 8E_s$ in the above expression gives the WER for the (8,4) extended Hamming code.

$$\text{WER} = 1 - \int_{0}^{\infty} \frac{1}{\sqrt{\pi N_o}} \exp\left[-\frac{1}{N_o}(x - \sqrt{8E_s})^2\right]\left[\int_{-x}^{x} \frac{1}{\sqrt{\pi N_o}} \exp\left(-\frac{y^2}{N_o}\right) dy\right]^{7} dx \qquad (4.2.2)$$

This can be simplified as:

$$\mathrm{WER} = 1 - \int_0^{\infty} \frac{1}{\sqrt{2\pi}}[1 - 2\,Q(x)]^7 \exp\left[-\frac{\left(x - \sqrt{8\,\gamma_b}\right)^2}{2}\right] dx \tag{4.2.3}$$

where γ_b is the signal-to-noise ratio per information bit.

Usually, it is very difficult to find an expression for the WER of a linear block code. Instead the WER can be upper bounded by using a union bound. That is,

$$\mathrm{WER} \le \sum_{m=2}^{2^k} P_m \tag{4.2.4}$$

where P_m is the probability of choosing codeword m when codeword 1 was transmitted. For codewords transmitted over an AWGN channel using BPSK modulation P_m equals:

$$P_m = Q\left(\sqrt{2 \cdot R \cdot w_m \frac{E_b}{N_o}}\right) \tag{4.2.5}$$

where R is the rate of the code and w_m is the weight of the mth codeword.

The (8,4) extended Hamming code has the following weight enumerator polynomial: $A(z) = 1 + 14\,z^4 + z^8$. Consequently, the WER can be bounded as:

$$\mathrm{WER} \le 14\,Q\left(\sqrt{4\frac{E_b}{N_o}}\right) + Q\left(\sqrt{8\frac{E_b}{N_o}}\right) \tag{4.2.6}$$

Since the (24,12) extended Golay code is a rate half code with the following weight enumerator polynomial: $A(z) = 1 + 759\,z^8 + 2576\,z^{12} + 759\,z^{16} + z^{24}$, the upper bound on its WER is:

$$\mathrm{WER} \le 759\,Q\left(\sqrt{8\frac{E_b}{N_o}}\right) + 2576\,Q\left(\sqrt{12\frac{E_b}{N_o}}\right) + 759\,Q\left(\sqrt{16\frac{E_b}{N_o}}\right) + Q\left(\sqrt{24\frac{E_b}{N_o}}\right) \tag{4.2.7}$$

For an AWGN channel, a correlation decoder will perform maximum likelihood decoding. This is done by comparing the noise corrupted information sequence to all codewords and choosing the closest (largest correlation) codeword. The correlation decoder can be used as a comparison to further verify that the sequential algorithm performs maximum likelihood decoding.

Figures 4.2.1 and 4.2.2 show the measured word error rates of the sequential algorithm when used to decode the (8,4) and (24,12) codes. The solid points represent simulated data points. The points are joined by straight line segments. The upper bounds are included for reference.

The number of errors made by the correlation decoder was equal to the number of errors made by the sequential decoder using the Fano metric. As well, the theoretical error performance of the (8,4) code matches with the simulated error performance of the sequential algorithm. The observation that the sequential algorithm (with the Fano metric) performs maximum likelihood soft decision decoding should be expected given Theorem 2.

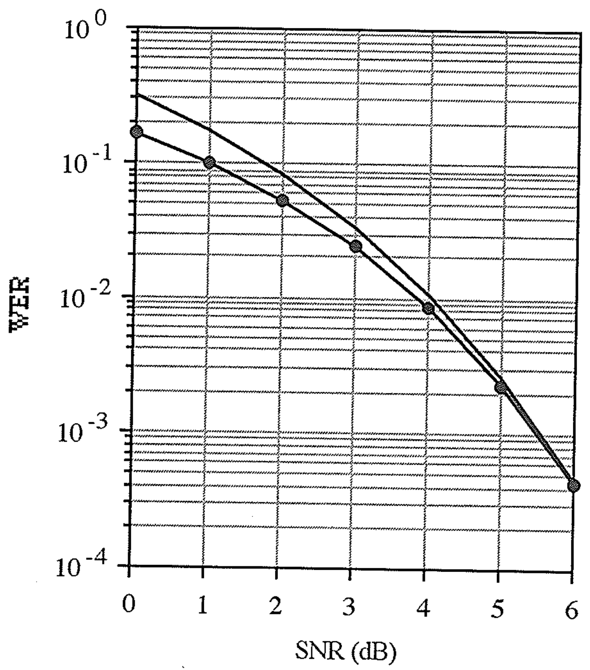


Figure 4.2.1. Error Performance of (8,4) Code

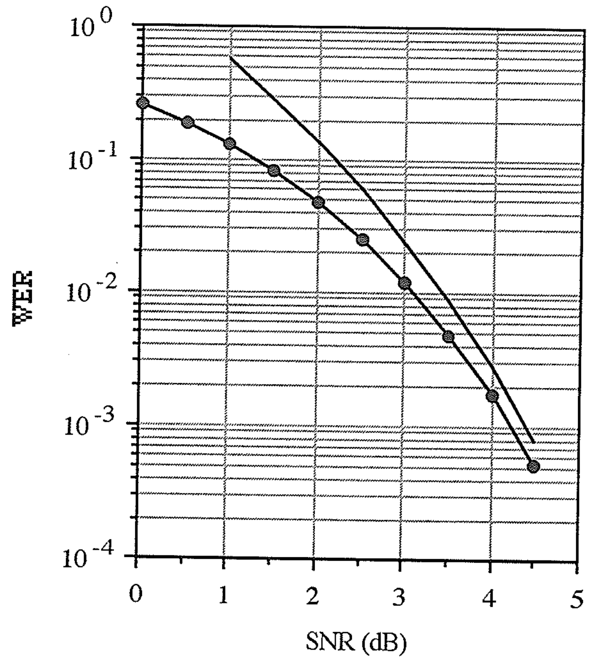


Figure 4.2.2. Error Performance of (24,12) Code

4.3 Complexity

Chapter 3 showed how to measure the computational complexity of the stack algorithm. In order to calculate the complexity, the number of nodes visited by the algorithm must be known. At the same time that the error performance was measured, the average number and average squared number of nodes visited as a function of signal-to-noise ratio were measured. Figures 4.3.1 and 4.3.2 show the average number of nodes visited for the (8,4) and (24,12) codes with the Fano metric. The vertical axis of the graphs show the average number of nodes, in excess of the minimum, that must be searched. For the (8,4) code, the minimum number of nodes that must be searched is 12, and, for the (24,12) Golay code, the minimum is 36 nodes. Figure 4.3.3 shows the average squared number of nodes visited for the (24,12) code. These exponentially decaying curves are quite similar to the average complexity curves for convolutional codes [ClCa81].

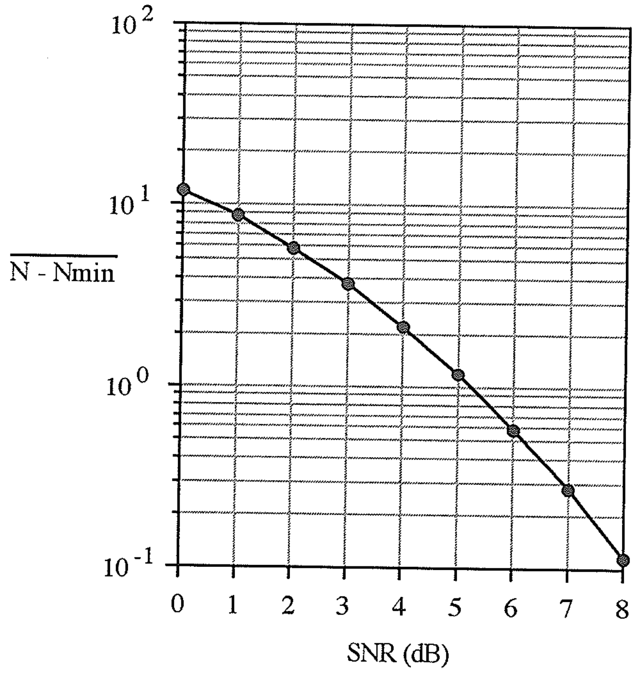


Figure 4.3.1. Average Nodes Searched for the (8,4) Code

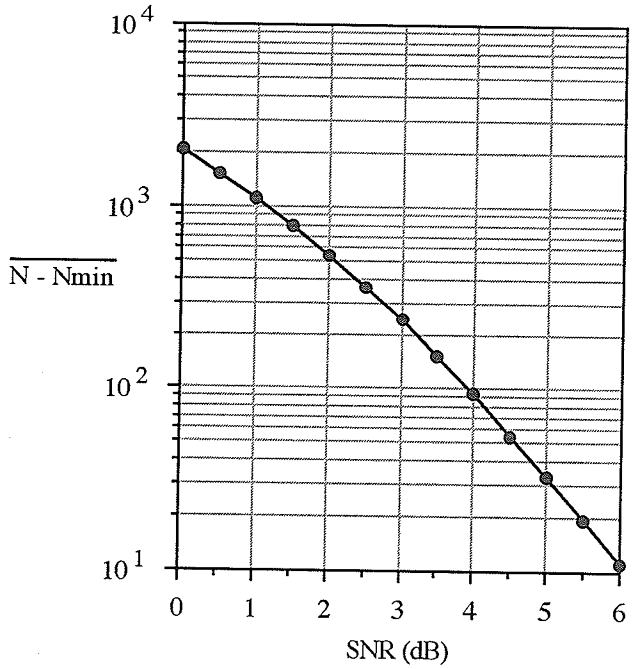


Figure 4.3.2. Average Nodes Searched for the (24,12) Code

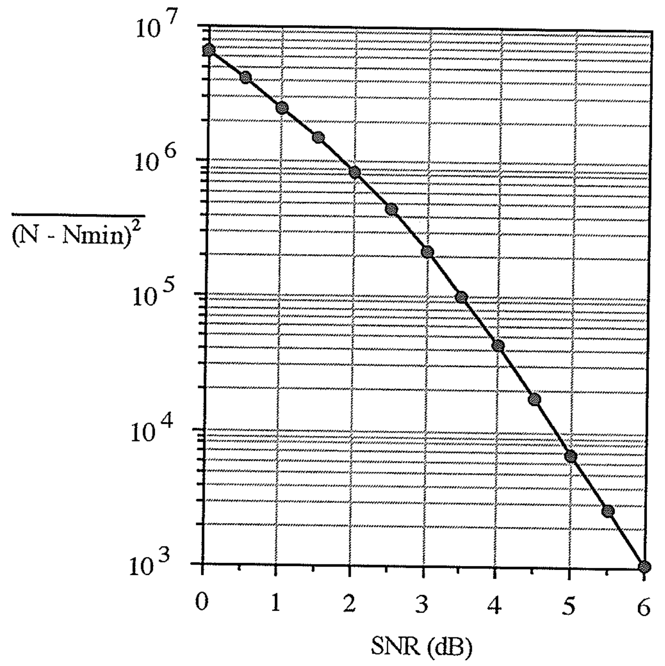


Figure 4.3.3. Average Squared Nodes Searched for the (24,12) Code

In certain situations, the variability in computation of the sequential algorithm can pose practical problems. The variance in number of nodes searched can be determined from the average and average squared nodes searched. That is,

$$\text{Variance} = \overline{N^2} - \overline{N}^2 \tag{4.3.1}$$

The variance for the (24,12) Golay code is plotted in Figure 4.3.4. The graph shows that the variance decreases with the signal-to-noise ratio. Consequently, any problems caused by variability are less hindering for high signal-to-noise ratios.

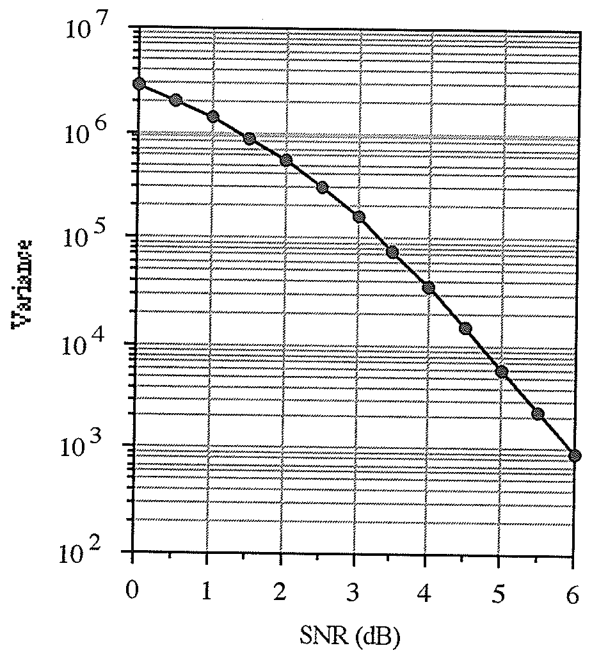


Figure 4.3.4. Variability in Nodes Searched for (24,12) Code

4.4 Comparisons

To judge the computational efficiency of the stack algorithm, comparisons are made to well known maximum likelihood decoding techniques. The comparisons are for the (24,12) extended Golay code only.

The conceptually simplest decoding technique is correlation decoding. For a correlation decoder the complexity to decode an (n,k) block code is $(n2^k\text{-}1)$ addition equivalent operations. This consists of $(n\text{-}1)2^k$ additions and $(2^k\text{-}1)$ comparisons. For the (24,12) Golay code, the complexity is $N_{CD} = 98303$ addition equivalent operations.

The Viterbi algorithm [Wolf78] is an improvement over a correlation decoder. The complexity of the Viterbi algorithm consists of N_m metric additions and N_c metric comparisons. The number of metric additions is equal to the number of branches in the trellis less two for the first two branches where additions are not required. The number of comparisons is equal to the number of nodes in the trellis with two entering branches. The calculation of N_m and N_c and, in turn, the total complexity is shown in Appendix 4-A. The results are given below.

$$N_m = 2^{n-k+1}(2k - n + 2) - 6 \tag{4.4.1}$$

$$N_c = 2^{n-k}(2k - n + 1) - 1 \tag{4.4.2}$$

$$N_{VA} = 2^{n-k}(6k - 3n + 5) - 7 \tag{4.4.3}$$

For the (24,12) Golay code, this amounts to $N_{VA} = 20473$ addition equivalent operations.

The complexity of Snyders and Be'ery's [SnBe89] algorithm for the (24,12) Golay code is 683 on average and has a minimum of 539 and a maximum of 827 addition equivalent operations. Recently, this has been bettered by Vardy and Be'ery [VaBe91]. Their algorithm has a maximum complexity of 651 addition equivalent operations. Other low complexity techniques are those developed by Conway and Sloane [CoSl86] requiring 1614 operations, Be'ery and Snyders [BeSn86] requiring 1551 (1159 on average) operations, and Forney [Forn88] requiring 1351 operations.

Given that the sequential algorithm visits N nodes the complexity is bounded by:

$$N_{SA} \leq \frac{1}{5}N^2 + N \tag{4.4.4}$$

and the average complexity is bounded by:

$$\overline{N_{SA}} \leq \frac{1}{5}\overline{N^2} + \overline{N} \tag{4.4.5}$$

By using this formula in conjunction with Figures 4.3.2 and 4.3.3, the complexity of the sequential algorithm can be bounded as a function of the signal-to-noise ratio. The following table lists the signal-to-noise ratio at which the sequential algorithm comparatively becomes the better algorithm for decoding the Golay code.

Technique	Maximum Complexity	SNR
Correlation Decoder	98303	2 dB
Viterbi Algorithm	20473	3 dB
Conway-Sloane (86)	1614	5 dB
Be'ery-Snyders (86)	1551	5 dB
Forney (88)	1351	5 dB
Snyders-Be'ery (89)	827	6 dB
Vardy-Be'ery (91)	651	6 dB

The table shows that whenever the signal power is at least four times larger than the noise power the sequential algorithm is the most efficient algorithm for decoding the (24,12) Golay code. As signal-to-noise ratios increase, the sequential algorithm quickly approaches its minimum complexity of 294 addition equivalent operations. Of course, the possibility always remains that the noise will cause the algorithm to search the entire code tree. In that unlikely situation the algorithm will have to perform 57342 metric additions and about 214 million metric comparisons.

4.5 Summary

Simulations were performed to measure the effectiveness of the sequential algorithm as a decoding alternative for linear block codes. As expected from the theory in chapter 2, the sequential algorithm performs maximum likelihood decoding. Using the complexity measures of chapter 3, the sequential algorithm has a favorable average complexity at moderate to high signal-to-noise ratios. In fact, by 3 dB it outperforms the Viterbi algorithm, and by 6 dB it is the most efficient algorithm for decoding the (24,12) Golay code.

Appendix 4-A Complexity of Viterbi Algorithm

To calculate N_m and N_c the trellis should be divided into three regions as shown below (for a (5,3) block code). Regions 1, 2 and 3 correspond to where the trellis grows, stays constant, and collapses, respectively. Assuming that the rate of the code is greater than one half, region 1 extends to a depth of (n - k) symbols, region 2 falls between the depths of (n - k) and k symbols, and region 3 contains all nodes at depths greater than k.

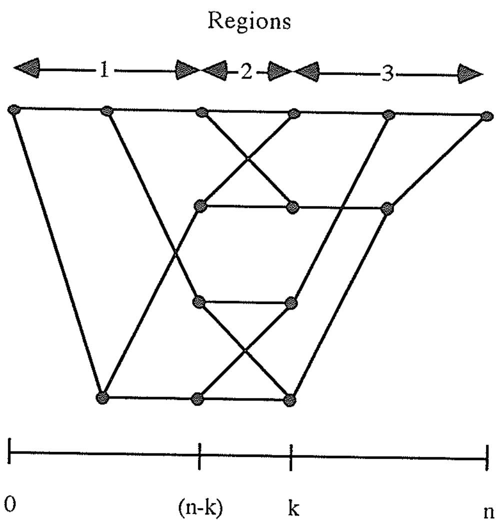


Figure 4.A.1. Trellis Regions

In region 1 the number of branches doubles for each step into the trellis. Therefore,

$$N_m^{(1)} = 2 + 4 + \cdots + 2^{n-k} = 2^{n-k+1} - 2 \tag{4.A.1}$$

In region 2 the trellis has a constant number of branches and nodes. Thus,

$$N_m^{(2)} = 2^{n-k+1}\,(\,k - (n-k)\,) = 2^{n-k+1}\,(2k - n) \tag{4.A.2}$$

In region 3 the number of branches halves as the trellis contracts. Hence,

$$N_m^{(3)} = 2^{n-k} + 2^{n-k+1} + \cdots + 2 = 2^{n-k+1} - 2 \tag{4.A.3}$$

The total number of metric additions is:

$$N_m = N_m^{(1)} + N_m^{(2)} + N_m^{(3)} - 2 = 2^{n-k+1}(2k - n + 2) - 6 \qquad (4.A.4)$$

Comparisons are only needed in regions 2 and 3 since two branches enter every node in these regions. The number of nodes in region 2 is:

$$N_c^{(2)} = 2^{n-k}(k - (n - k)) = 2^{n-k}(2k - n) \qquad (4.A.5)$$

The number of nodes in region 3 halves for each step into the trellis so:

$$N_c^{(3)} = 2^{n-k-1} + 2^{n-k-2} + \dots + 1 = 2^{n-k} - 1 \qquad (4.A.6)$$

Adding gives the total number of comparisons:

$$N_c = N_c^{(2)} + N_c^{(3)} = 2^{n-k}(2k - n + 1) - 1 \qquad (4.A.7)$$

Therefore, the total complexity of the Viterbi algorithm is:

$$N_{VA} = N_m + N_c = 2^{n-k}(6k - 3n + 5) - 7 \qquad (4.A.8)$$

equivalent real number additions.

Chapter 5 - Suboptimum Implementations of the SSA

It has been shown that the stack algorithm can be applied to block codes once a tree or trellis is established. Using the Fano metric, the stack algorithm can perform maximum likelihood soft decision decoding. As well, the computational complexity of the algorithm decreases quickly with an increasing signal-to-noise ratio. Therefore, the stack algorithm is a viable candidate for soft decision decoding of linear block codes at moderate to high signal-to-noise ratios.

There are several unattractive features of the stack algorithm when applied to convolutional codes. These are unbounded computation, input buffer overflow, stack overflow, stack sort, and path storage. When the algorithm is used for block codes some of these problems no longer exist or are alleviated.

First of all, sequential decoding should not be used with convolutional codes that have rates above the computational cutoff rate of the channel. In other words, the sequential algorithm should only be used when the signal-to-noise ratio is sufficiently high to push the computational cutoff rate above the code rate. The reason for this is that the average number of computations becomes unbounded (infinite) at rates above the computational cutoff rate. On the other hand, for block codes, only a finite number of computations (e.g. maximum of 4^k for an (n,k) binary block code) can be performed before decoding is completed.

The unbounded computation leads to a second problem which is input buffer overflow. That is, the decoding is not finished before new data arrives that fills and eventually overflows a finite input buffer. As a result, data is lost causing future decoding errors. This problem can be controlled for block codes by allotting a maximum number of nodes that the algorithm can visit before proceeding to the next codeword (see section 5.1).

A third problem related to the problem of unbounded computation is stack overflow.

Too many decoding steps will add too many entries to the stack with the result that the stack fills and overflows. Further entries to the full stack will either be, or cause other entries to be, lost resulting in degraded error performance. For block codes there are only q^k paths that the algorithm can possibly visit. Since the head of each path is stored in the stack, the stack will contain q^k entries at most. Therefore, the stack will never overflow for block codes provided that it can hold q^k entries. Unfortunately, the stack size becomes prohibitively large as the number of information symbols increases. Therefore a smaller stack must be used which, in turn, has a probability of overflowing. In section 5.2 the effects of a reduced stack size are described.

The fourth problem with the stack algorithm is the computationally burdensome stack sort needed to determine the partial path with the best metric. The priority queue solves this problem for both convolutional and block codes.

Finally, when the Viterbi algorithm or the sequential algorithm is used with convolutional codes the length of the explored path can grow quite large. Therefore, the decoder has to be able to manipulate long and variable length paths. For (n,k) block codes the maximum path length is n symbols. Therefore, the decoder can be designed to work with paths of this length and the decoder can output n symbols at a time as a codeword is decoded.

5.1 Reducing Computational Complexity

In order to reduce the complexity a scheme where the algorithm is limited to a maximum number of branches or nodes that it can search can be used. If the codeword is not decoded after L branches are searched, then the decoder can either output hard decisions on the individual bits, force the algorithm through the trellis (i.e. no backtracking), or output an erasure symbol.

If this maximum number of nodes, L, happens to be less than q^k then not only will the computational complexity be reduced but the priority queue size can also be reduced. Since L nodes are only visited there will only be L insertions into the priority queue (one insertion per node visited) and hence the queue will only need to hold less than L entries. Using this scheme for an (n,k) block code over GF(q) limits the stack to:

$$\text{maximum queue size} \le \min\left[\, q^k, L \,\right] \qquad (5.1.1)$$

The computational complexity will be reduced since L will be less than the maximum number of nodes the algorithm can possibly visit. Without limiting the number of nodes the average complexity of the sequential algorithm is given by:

$$\overline{N_{SA}} \approx \overline{N} + \frac{q-1}{3q-1}\overline{N^2} = \sum_{N=N_{min}}^{N_{max}} \left(N + \frac{q-1}{3q-1} N^2 \right) \Pr[\, N \,] \qquad (5.1.2)$$

where

$$N_{min} = qk + (n-k) \qquad (5.1.3)$$

$$N_{max} = \left(n - k + \frac{q}{q-1} \right) q^k - \frac{q}{q-1} \qquad (5.1.4)$$

and N is the number of nodes visited.

The average complexity with limited searching is:

$$\overline{N_{SA}}(L) \approx \left(L + \frac{q-1}{3q-1} L^2 \right) \Pr\left[\, N \ge L \,\right] + \sum_{N=N_{min}}^{L-1} \left(N + \frac{q-1}{3q-1} N^2 \right) \Pr[\, N \,] \quad (5.1.5)$$

which is clearly less than the average complexity with full searching since $L < N_{max}$.

The following theorem can be used as a guide to choose a search limit. It should be realized that limiting the amount of searching will invariably reduce the performance of the decoder as determined by the WER.

Theorem 3

A sequential decoder must be allowed to visit a minimum number of nodes in order to guarantee that errors due to insufficient decoding occur less frequently than errors due to channel noise. When a binary block code is transmitted over an AWGN channel using BPSK modulation, a soft-decision decoder must be allowed to visit a minimum number of nodes, L, that is lower bounded as:

$$L \geq N_{min}\, 2^{R \cdot d_{min}}$$

N_{min} *is the minimum number of nodes that must be visited to decode a codeword, R is the code rate, and* d_{min} *is the minimum distance of the code.*

The attractiveness of this theorem is that the result is independent of the signal-to-noise ratio. Intuitively, this can be explained by realizing that at low signal-to-noise ratios the word error rate is already high so errors due to insufficient searches can also be large so the ordinarily large searches can be reduced. At high signal-to-noise ratios the error rate is low so the algorithm must allow for large searches in order to keep decoding failures small. Two proofs of this theorem are provided in Appendix 5-A.

Using the previous theorem as a guide, simulations were performed to determine the degradation in error performance with limited searching. The simulations were performed for the (24,12) extended Golay code transmitted over an AWGN channel with BPSK modulation. This code has a minimum distance of 8 and consequently, by the above theorem, the minimum number of nodes to be searched is 576. The search limits used in the simulations were 500, 1000, 2000, and 4000 nodes.

The following graph shows the WER as a function of signal-to-noise ratio for the four search limits and for the unlimited search. The lower curve is for the unlimited searched, and the curves move up as the search limit is reduced.

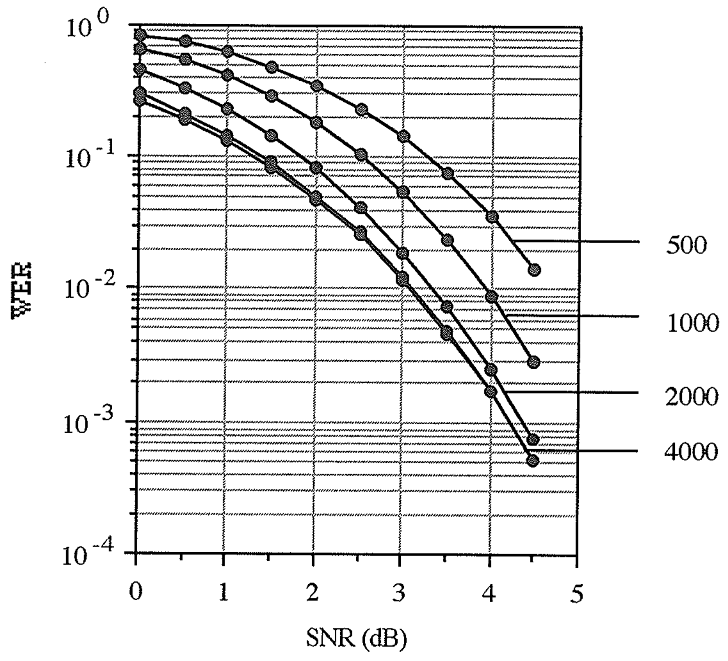


Figure 5.1.1. WER with Search Limits of 4000, 2000, 1000, and 500 Nodes. (Lower curve represents an unlimited search)

The simulations show that limiting the search to 4000 nodes has a negligible effect on the error performance of the decoder. A small degradation in the WER is noticeable for search limits of 1000 and 2000 nodes. For these limits, the loss in error performance is less than 1 dB. Reducing the search limit further results in greater losses in error performance.

For the (24,12) Golay code, the maximum number of nodes that can be visited by the stack algorithm is 57342 nodes. The simulations show that a limit of about 4000 nodes is sufficient to guarantee essentially optimal error performance. This represents a decrease of over an order of magnitude.

Since the number of comparisons increases with the square of the number of nodes searched, the decrease in nodes searched results in a decrease of over two orders of magnitude in the maximum number of comparisons. Consequently, the variation in computational complexity of the sequential algorithm can be reduced by a factor of 100 for the (24,12) Golay code by limiting the search to 4000 nodes.

As well as measuring the degradation in error performance, the probability that a search reaches its limit was measured. This is shown below for four search limits, L, where Pr [N ≥ L] is the probability that the number of nodes searched, N, exceeds L. The lower curve is for a limit of 4000 nodes while the upper curve is for 500 nodes.

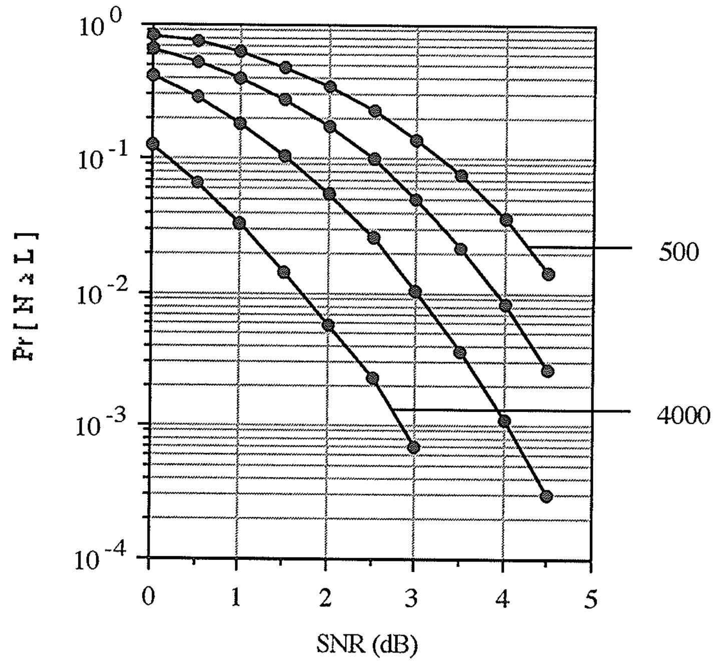


Figure 5.1.2. Pr [N ≥ L] for Search Limits of 4000, 2000, 1000, and 500 Nodes.

These simulations show that the probability that the number of nodes exceeds some search limit decays exponentially with the signal-to-noise ratio. Thus, the number of errors introduced by limiting the search will decrease exponentially with the signal-to-noise ratio. Since the WER with limited searching is composed of errors due to noise and errors due to insufficient searching, the search limit should be selected to guarantee that Pr [N ≥ L] decays as fast as or faster than the WER without limited searching. This can be shown by using a union bound. First, the WER with limited searching is:

$$\text{WER(L)} = \Pr[\text{ error due to noise OR search exceeds limit }] \qquad (5.1.6)$$

where WER(L) designates the WER with searching limited to L nodes.
Using the union bound, this can be approximated as:

$$\text{WER(L)} \approx \text{WER} + \Pr\left[N \geq L \right] \qquad (5.1.7)$$

for high signal-to-noise ratios, where

$$\text{WER} = \Pr[\text{ error due to noise }] \qquad (5.1.8)$$

and

$$\Pr\left[N \geq L \right] = \Pr[\text{ search exceeds limit }] \qquad (5.1.9)$$

This shows that in order to guarantee the same rate of decay with signal-to-noise ratio for the WER(L) the probability of exceeding the search limit must decay at a rate greater than or equal to the rate of decay of the WER. This is the criteria used in Appendix 5-A to prove the theorem for choosing a search limit.

For the (24,12) Golay code, the probability that the number of nodes searched is greater than or equal to 4000 is much smaller and decays much faster than the WER. Consequently, this search limit should have a negligible effect on the overall WER. As previously noted, this observation is confirmed by the simulations.

5.2 Reducing Hardware Complexity

The second problem with the optimum implementation of the sequential stack algorithm for block codes is that the required stack size grows exponentially with the number of information symbols in the code. That is, the required stack size for optimum decoding of an (n,k) block code over GF(q) is q^k. Therefore, in order to use the stack algorithm for one of these larger codes the optimal stack size has to be sacrificed in favor of a more practical size. Since the decoder uses only a small portion of the entire stack under reasonable noise levels - a complete stack is only needed when the decoder visits every path (i.e. considers every codeword) - reducing the stack size should be feasible.

To determine the effect of a reduced stack on the error performance of the decoder simulations were performed. Again, the simulations were for the (24,12) extended Golay code transmitted over an AWGN channel using BPSK modulation. This code optimally requires a stack with 4096 elements. The simulations were for stack sizes of 100, 40 and 10 elements. The WER as a function of signal-to-noise ratio is shown in the next figure for these three cases as well as the optimum WER (lower curve).

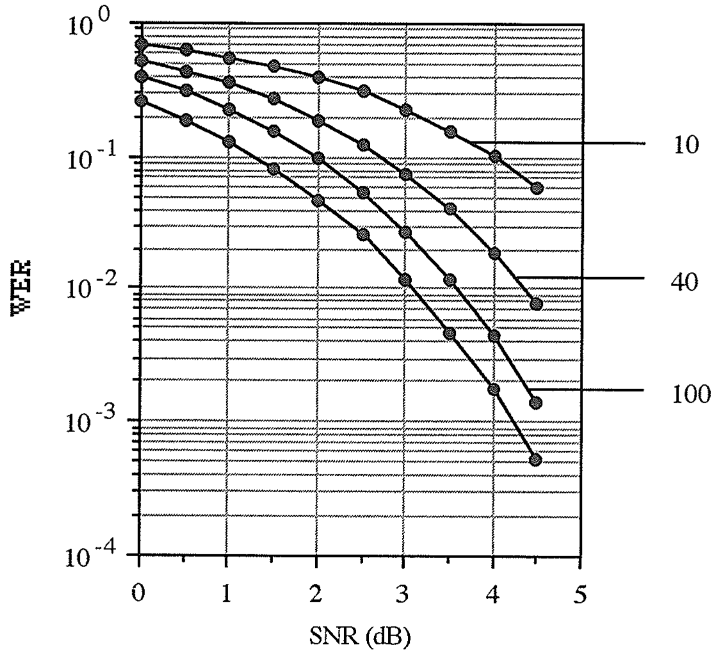


Figure 5.2.1. WER with Stacks of 100, 40, and 10 Elements. (Lower curve represents an optimum stack)

The simulations show that a stack of 100 elements does not degrade the performance of the sequential algorithm significantly. The degradation is less than 0.5 dB in this simulation. This represents a stack reduction of about 97.5 percent. Further reductions in the stack size result in larger degradations in error performance.

The side effect of a reduced stack is a reduced computational complexity. Since all adjacent elements in the stack are compared after every insertion and deletion, the number of comparisons is directly proportional to the number of elements in the stack. Therefore, a smaller stack size translates into less metric comparisons in the priority queue.

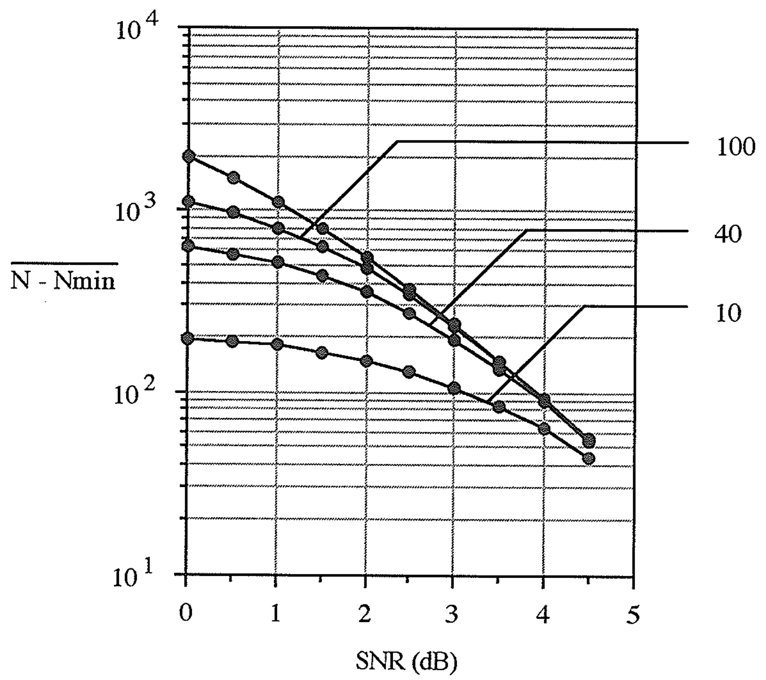


Figure 5.2.2. Average Nodes Searched for Stacks of 100, 40, and 10 Elements. (Upper curve represents an optimum stack)

The number of nodes searched as a function of signal-to-noise ratio is plotted in the above graph for the three stack sizes along with the optimum stack size. The simulations show that the algorithm visits less nodes with a reduced stack. Since the number of nodes visited is reduced, this will translate into less metric additions and less metric comparisons performed by the decoder.

Thus, reducing the stack size has a doubly positive effect on the computational complexity. If the algorithm were to visit the same number of nodes as it would with a complete stack then the complexity would be smaller since less comparisons are made in the stack. The reduced stack causes the algorithm to visit less nodes, though. Consequently, even less metric additions and comparisons need to be performed by the algorithm. Hence reducing the stack size not only reduces the hardware complexity but reduces the computational complexity.

5.3 Summary

Two obstacles to the practical implementation of the sequential stack algorithm for block codes are the variability in computational complexity and the stack (priority queue) size. The variation in computational complexity can be reduced by limiting the length of the search performed by the sequential algorithm. Since a full stack is only required for the rare occasion (at moderate to high signal-to-noise ratios) when the decoder considers a large number of potential codewords, the hardware complexity (stack size) can be dramatically decreased.

Unfortunately, both of these techniques come at the expense of error performance, though, this might be a small price to pay for the reductions in complexity. Fortunately, both techniques have the added advantage that they reduce both computational and hardware complexity.

It must be remembered, though, that these conclusions are based to a large degree on observations of the techniques as applied to a single code. While not anticipated, the gains may not be as sparkling for other block codes. Unfortunately, it is difficult to precisely predict the effects of any complexity reduction technique used with sequential decoders. In fact, most aspects of sequential decoders can not be predicted accurately. This was noted by Wozencraft and Jacobs very early in the study of sequential decoding algorithms:

> *"[The] analytical difficulties with sequential decoding are such that even the tightest bounds that have been derived are not numerically accurate enough for the purposes of engineering design." [WoJa65]*

Appendix 5-A-1 First Proof of Theorem

Theorem

A sequential decoder must be allowed to visit a minimum number of nodes in order to guarantee that errors due to insufficient decoding occur less frequently than errors due to channel noise. When a binary block code is transmitted over an AWGN channel using BPSK modulation, a soft-decision decoder must be allowed to visit a minimum number of nodes, L, that is lower bounded as:

$$L \geq N_{min}\, 2^{R \cdot d_{min}}$$

N_{min} is the minimum number of nodes that must be visited to decode a codeword, R is the code rate, and d_{min} is the minimum distance of the code.

Before proving the theorem a result about the Gallager function must be shown. For a symmetric discrete memoryless channel (DMC) the Gallager function is given by [ClCa81]:

$$E_0(\rho) = -\log_2 \sum_{j=0}^{J-1} \left[\sum_{k=0}^{K-1} \frac{1}{K} [\Pr(j \mid k)]^{1/1+\rho} \right]^{1+\rho} \qquad (5.A.1)$$

where K and J are the number of channel inputs and outputs, respectively, and Pr (j | k) are the channel transition probabilities.

The DMC Gallager function can be extended to an unquantized channel by replacing the summation over the channel outputs by an integral and replacing the channel transition probabilities by the conditional channel output density functions, p (r | k). That is,

$$E_o(\rho) = -\log_2 \int_{-\infty}^{\infty} \left[\sum_{k=0}^{K-1} \frac{1}{K} [\, p(r \mid k) \,]^{1/1+\rho} \right]^{1+\rho} dr \qquad (5.A.2)$$

Since [ViOm79],

$$\frac{\partial E_o(\rho)}{\partial \rho} > 0, \qquad \rho \geq 0 \qquad (5.A.3)$$

the Gallager function is a non-decreasing function for positive values of ρ and is upper bounded by its asymptotic value as ρ tends to infinity. That is,

$$E_o(\rho) \leq \lim_{\rho \to \infty} E_o(\rho) \equiv E_o(\infty), \qquad \rho \geq 0 \qquad (5.A.4)$$

where

$$E_o(\infty) = -\log_2 \int_{-\infty}^{\infty} \left[\prod_{k=0}^{K-1} p(r \mid k) \right]^{1/K} dr \qquad (5.A.5)$$

as shown in Appendix 5-B.

For an AWGN channel with BPSK modulation K = 2 and:

$$p(r \mid k) = \frac{1}{\sqrt{\pi N_o}} \exp\left[- \frac{(r - (2k-1)\sqrt{E_s})^2}{N_o} \right] \qquad (5.A.6)$$

Substituting and simplifying,

$$E_o(\infty) = \frac{R \cdot \gamma_b}{\ln 2} \qquad (5.A.7) \qquad \text{and} \qquad E_o(\rho) \le \frac{R \cdot \gamma_b}{\ln 2} \qquad (5.A.8)$$

where R is the code rate and γ_b is the signal-to-noise ratio per information bit.

Armed with the bound on the Gallager function the theorem can be proved. The WER can be approximated by:

$$\text{WER} \approx A\, Q\left(\sqrt{2 \cdot R \cdot d_{min} \frac{E_b}{N_o}} \right) \approx \frac{A}{2} \exp\left[- R \cdot d_{min} \cdot \gamma_b \right] \qquad (5.A.9)$$

for high signal-to-noise ratios.

In order to choose a limit on the number of nodes the sequential algorithm should be allowed to visit, the distribution of the number of nodes visited should be known. A typical plot of this distribution can be found in [Sava66] or [LiCo83]. The pareto like distribution is characteristic of sequential algorithms.

The Pr [N ≥ L] curve is bounded by:

$$\left(\frac{L}{N_{min}} \right)^{-\rho} \le \Pr\left[N \ge L \right] \le B \left(\frac{L}{N_{min}} \right)^{-\rho} \qquad (5.A.10)$$

where B ≥ 1.

Approximating Pr [N ≥ L] by the upper bound and rewriting gives:

$$\Pr\left[N \ge L \right] \approx B \left(\frac{L}{N_{min}} \right)^{-\rho} = B \exp\left[-\rho \ln \left(\frac{L}{N_{min}} \right) \right] \qquad (5.A.11)$$

For Pr [N ≥ L] to decay as the WER the following equality must hold:

$$\rho \ln\left(\frac{L}{N_{min}}\right) = R \cdot d_{min} \cdot \gamma_b \tag{5.A.12}$$

or

$$\ln\left(\frac{L}{N_{min}}\right) = \frac{R \cdot d_{min} \cdot \gamma_b}{\rho} \tag{5.A.13}$$

The pareto exponent ρ is the implicit solution of:

$$R = \frac{E_o(\rho)}{\rho} \tag{5.A.14}$$

where R is the code rate and $E_o(\rho)$ is the Gallager function.

The bound on the Gallager function, bounds the pareto exponent ρ as:

$$\rho = \frac{E_o(\rho)}{R} \leq \frac{\frac{R \cdot \gamma_b}{\ln 2}}{R} = \frac{\gamma_b}{\ln 2} \tag{5.A.15}$$

Substituting the bound for ρ gives:

$$\ln\left(\frac{L}{N_{min}}\right) \geq \frac{R \cdot d_{min} \cdot \gamma_b}{\frac{\gamma_b}{\ln 2}} = R \cdot d_{min} \cdot \ln 2 = \ln 2^{R \cdot d_{min}} \tag{5.A.16}$$

or

$$L \geq N_{min} \, 2^{R \cdot d_{min}} \tag{5.A.17}$$

which proves the theorem.

Appendix 5-A-2 Second Proof of Theorem

The second proof uses a result from Viterbi and Omura [ViOm79] for decoding convolutional codes. When sequentially decoding convolutional codes the number of branch extensions per node can be limited to:

$$\hat{N}_m = \frac{L}{N_{min}} = \text{branch extensions per node} = 2^{k(M+1)} \qquad (5.A.18)$$

for an (n,k) constraint length (M + 1) convolutional code. This will guarantee that errors due to insufficient searching have the same exponential decay with signal-to-noise ratio as errors due to noise.

Using the following bound on a convolutional code's free distance:

$$d_f \leq n\,(M+1) \qquad (5.A.19)$$

lowers bound the number of branch extensions per node to:

$$\hat{N}_m = \frac{L}{N_{min}} \geq 2^{R \cdot d_f} \qquad (5.A.20)$$

where R is the code rate.

Replacing d_f by d_{min} (the minimum distance of a block code) and multiplying by the minimum number of nodes the sequential algorithm must visit to decode a linear block code gives:

$$L \geq N_{min}\, 2^{R \cdot d_{min}} \qquad (5.A.21)$$

which proves the theorem.

Appendix 5-B Proof of Gallager Function Limit

The Gallager function for an unquantized memoryless channel is defined as:

$$E_o(\rho) = -\log_2 \int_{-\infty}^{\infty} \left[\sum_{k=0}^{K-1} \frac{1}{K} [\, p(r \mid k) \,]^{1/1+\rho} \right]^{1+\rho} dr \qquad (5.B.1)$$

Therefore, the limit as ρ tends to infinity is:

$$E_o(\infty) \equiv \lim_{\rho \to \infty} E_o(\rho) = \lim_{\rho \to \infty} -\log_2 \int_{-\infty}^{\infty} \left[\sum_{k=0}^{K-1} \frac{1}{K} [\, p(r \mid k) \,]^{1/1+\rho} \right]^{1+\rho} dr \qquad (5.B.2)$$

which can be rewritten as:

$$E_o(\infty) = -\log_2 \int_{-\infty}^{\infty} \exp\left[\lim_{\rho \to \infty} \frac{\ln \sum_{k=0}^{K-1} \frac{1}{K} [\, p(r \mid k) \,]^{1/1+\rho}}{\frac{1}{1+\rho}} \right] dr \qquad (5.B.3)$$

Using l'Hospital's rule this becomes:

$$E_o(\infty) = -\log_2 \int_{-\infty}^{\infty} \exp\left[\sum_{k=0}^{K-1} \frac{1}{K} \ln p(r \mid k) \right] dr \qquad (5.B.4)$$

or

$$E_o(\infty) = -\log_2 \int_{-\infty}^{\infty} \left[\prod_{k=0}^{K-1} p(r \mid k) \right]^{1/K} dr \qquad (5.B.5)$$

which is the desired result.

Chapter 6 - Conclusion

Unlike convolutional codes, there are no general algorithms for efficient maximum likelihood soft decision decoding of linear block codes. Instead of trying to find such an algorithm, though, it might be easier to adapt a convolutional algorithm to a block code. Perhaps, an even easier solution is to make a block code look like a convolutional code.

In chapter 2 it was shown that all block codes have a tree representation and all linear block codes also have a trellis representation. This being so, it should be possible to use the Viterbi algorithm or a sequential algorithm to decode a linear block code. Furthermore, instead of trying to suit the algorithms to the block code it is easier to design block code encoders suited to the algorithms. One encoder realization treats a block code as if it were a rate one time-varying convolutional code. If the block code is cyclic, then the convolutional code need no longer be time-varying. Using the convolutional encoder realization, the sequential algorithm (with the Fano metric) is able to perform maximum likelihood decoding of linear block codes.

Chapter 3 dealt with the issue of computational complexity. Normally, the complexity of the sequential algorithm is measured in terms of the average amount of searching that it performs. For block codes, though, the complexity is normally measured in terms of equivalent real number additions. Chapter 3 gives an expression to translate the measured complexity (average nodes searched) into the complexity (equivalent real number additions) for block codes. The complexity of the sequential algorithm can be summarized as follows: the number of additions grows linearly with the number of nodes searched while the number of comparisons grows quadratically with the number of nodes searched. Consequently, too much searching will make the sequential algorithm inefficient. Therefore the algorithm is best used at moderate to high signal-to-noise ratios when the amount searching is small.

Chapter 4 gives a summary of computer simulations of the sequential algorithm. The simulations confirmed that the sequential algorithm is maximum likelihood. As well, the simulations showed that the algorithm is computationally efficient for moderate signal-to-noise ratios. In fact, by 6 dB, the sequential algorithm is the most efficient algorithm for decoding the (24,12) Golay code.

Finally, chapter 5 discussed some refinements that can be made to the sequential algorithm to reduce its complexity. These included setting an upper limit to the number of nodes the algorithm can search, and reducing the size of the stack that holds all the contending paths. These refinements come at the cost of error performance.

To conclude, the sequential algorithm is definitely a viable alternative for decoding linear block codes. All that is required is that the block code be made to look like a convolutional code. Once that step is done, any convolutional decoding algorithm, including the sequential algorithm, can be applied easily.

The convolutional code approach to linear block codes is one of the three key contributions of this thesis. The second contribution is the observation and proof that the sequential algorithm is able to perform maximum likelihood decoding when the metric satisfies certain conditions. Finally, the complexity measures of chapter 3, together, form the last of the three key contributions of the thesis.

6.1 Recommendations for Further Study

It may be interesting to pursue the following ideas:

1. In 1979, Solomon and van Tilborg coauthored a paper that showed how to transform a rate 1/n quasi-cyclic code into a rate 1/n convolutional code [SoVa79]. This approach requires that the encoder be pre-loaded with the end of the information

sequence. Unfortunately, when an information sequence has to be decoded, the end of the sequence is not known with certainty. In its current form, the sequential algorithm can not be used successfully when the initial encoder state is unknown. Modifying the sequential algorithm to suit this encoder is an open problem.

2. By casting a block code in the form of a convolutional code, perhaps time-varying, it has been possible to borrow convolutional decoding algorithms for the purpose of decoding block codes. There may be other results for convolutional codes that can be extended to block codes (i.e. rate one convolutional codes).

3. For convolutional codes, the sequential algorithm has a pareto distribution of computation. This distribution has been used to determine the signal-to-noise ratio at which the average number of nodes searched by the sequential algorithm becomes unbounded. For block codes, due to the finite tree or trellis, the average number of nodes searched will never become unbounded. A more useful measure for a block code might be the signal-to-noise ratio at which the complexity of the sequential algorithm is worse than the Viterbi algorithm. For example, this occurred at 3 dB for the (24,12) Golay code. So far, no results have been developed to predict the signal-to-noise ratio at which this would occur.

4. The memory between successive trees for concatenated codes may cause practical problems. This can be avoided by clearing the contents of the convolutional encoder. A better solution may be to ignore the memory and determine the degradation in error performance caused by this assumption.

References

[AnMo91]	Anderson, J.B., and Mohan, S., *Source and Channel Coding: An Algorithmic Approach*, Kluwer Academic Publishers, 1991.
[BeSn86]	Be'ery, Y., and Snyders, J., "Optimal Soft Decision Block Decoders Based on Fast Hadamard Transform," IEEE Transactions on Information Theory, vol. IT-32, No. 3, May 1986, pp. 355-364.
[Blah83]	Blahut, R.E., *Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Company, 1983.
[Chas72]	Chase, D., "A Class of Algorithms for Decoding Block Codes With Channel Measurement Information," IEEE Transactions on Information Theory, vol. IT-18, No. 1, January 1972, pp. 170-182.
[ChYa86]	Chang, C.Y., and Yao, K., "Systolic Array Architecture for the Sequential Stack Decoding Algorithm," SPIE vol. 696 Advanced Algorithms and Architectures for Signal Processing, 1986, pp. 196-203.
[ClCa81]	Clark, G.C. and Cain, J.B., *Error-Correction Coding for Digital Communications*, Plenum Press, 1981.
[CoSl86]	Conway, J.H. and Sloane, N.J.A., "Decoding Techniques for Codes and Lattices, Including the Golay Code and the Leech Lattice," IEEE Transactions on Information Theory, vol. IT-32, No. 1, January 1986, pp. 41-50.
[Forn66]	Forney, G.D. Jr., "Generalized Minimum Distance Decoding," IEEE Transactions on Information Theory, vol. IT-12, No. 2, April 1966, pp. 125-131.
[Forn88]	Forney, G.D. Jr., "Coset Codes II: Binary Lattices and Related Codes," IEEE Transactions on Information Theory, vol. IT-34, No. 5, September 1988, pp. 1152-1187.
[HeNo88]	Herro, M.A., and Nowack, J.M., "Simulated Viterbi Decoding Using Importance Sampling," IEE Proceedings, vol. 135, No. 2, April 1988, pp. 133-142.
[LiCo83]	Lin, S., and Costello, D.J., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Inc., 1983.
[MaMo82]	Matis, K.R., and Modestino, J.W., "Reduced-Search Soft-Decision Trellis Decoding of Linear Block Codes," IEEE Transactions on Information Theory, vol IT-28, No. 2, March 1982, pp. 349-355.
[Mass78]	Massey, J.L., "Foundations and Methods of Channel Coding," Proceedings of the International Conference on Information Theory and Systems, vol. 65, NTG-Fachberichte, September 1978.

[OfPe91] Offer, E., and Perkins, M.G., "Soft Decision Decoding of Block Codes and Concatenated Block-Convolutional Codes Using the Stack Algorithm," Globecom 91, pp. 765-769.

[PFTV88] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recipes in C - The Art of Scientific Computing*, Cambridge University Press, 1988.

[Proa89] Proakis, J.G., *Digital Communications*, 2nd ed., McGraw-Hill Book Company, 1989.

[Sava66] Savage, J.E., "Sequential Decoding - The Computation Problem," The Bell System Technical Journal, January 1966, pp. 149-175.

[SiBa54] Silverman, R.A., and Balser, M., "Coding for Constant-Data-Rate Systems - Part I. A New Error-Correcting Code," Proceedings of the IRE, vol. 42, September 1954, pp. 1428-1435.

[SnBe89] Snyders, J., and Be'ery, Y., "Maximum Likelihood Soft Decoding of Binary Block Codes and Decoders for the Golay Codes," IEEE Transactions on Information Theory, vol IT-35, No. 5, September 1989, pp. 963-975.

[SoVa79] Solomon, G., and van Tilborg, H.C.A., "A Connection Between Block and Convolutional Codes," SIAM Journal of Applied Math, vol. 37, No. 2, October 1979, pp. 358-369.

[VaBe91] Vardy, A., and Be'ery, Y., "Even More Efficient Soft Decoding of the Golay Codes," Proc. IEEE ISIT, Budapest, Hungary, June 24-28, 1991, p. 190.

[VaTr69] Van Trees, H.L., *Detection, Estimation, and Modulation Theory*, John Wiley and Sons, Inc., 1969.

[ViOm79] Viterbi, A.J. and Omura, J.K., *Principles of Digital Communications and Coding*, McGraw-Hill, New York, 1979.

[WoJa65] Wozencraft, J.M. and Jacobs, I.M., *Principles of Communication Engineering*, John Wiley and Sons, Inc., New York, 1965, p. 444.

[Wolf78] Wolf, J.K., "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis," IEEE Transactions on Information Theory, vol. IT-24, No. 1, January 1978, pp. 76-80.