Lossless Compression of Voiceband Data Signals

by:

Kent W. Ng

A thesis submitted to the
Faculty of Graduate Studies
in partial fulfillment of the requirements
for the degree of

Master of Science

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, 1996

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I also authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research

Kent Ng, 1996.

ISBN 0-612-13401-6

Name _____

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

Engineering, Electronics and Electrical
SUBJECT TERM

0544
SUBJECT CODE

UMI

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

Nom _____

Dissertation Abstracts International et *Masters Abstracts International* sont organisés en catégories de sujets. Veuillez s.v.p. choisir le sujet qui décrit le mieux votre thèse et inscrivez le code numérique approprié dans l'espace réservé ci-dessous.

UMI

_____ SUJET

CODE DE SUJET

Catégories par sujets

HUMANITÉS ET SCIENCES SOCIALES

COMMUNICATIONS ET LES ARTS

Architecture 0729
Beaux-arts 0357
Bibliothéconomie 0399
Cinéma 0900
Communication verbale 0459
Communications 0708
Danse 0378
Histoire de l'art 0377
Journalisme 0391
Musique 0413
Sciences de l'information 0723
Théâtre 0465

ÉDUCATION

Généralités 515
Administration 0514
Art 0273
Collèges communautaires 0275
Commerce 0688
Économie domestique 0278
Éducation permanente 0516
Éducation préscolaire 0518
Éducation sanitaire 0680
Enseignement agricole 0517
Enseignement bilingue et multiculturel 0282
Enseignement industriel 0521
Enseignement primaire. 0524
Enseignement professionnel 0747
Enseignement religieux 0527
Enseignement secondaire 0533
Enseignement spécial 0529
Enseignement supérieur 0745
Évaluation 0288
Finances 0277
Formation des enseignants 0530
Histoire de l'éducation 0520
Langues et littérature 0279
Lecture 0535
Mathématiques 0280
Musique 0522
Orientation et consultation 0519
Philosophie de l'éducation 0998
Physique 0523
Programmes d'études et enseignement 0727
Psychologie 0525
Sciences 0714
Sciences sociales 0534
Sociologie de l'éducation 0340
Technologie 0710

LANGUE, LITTÉRATURE ET LINGUISTIQUE

Langues
- Généralités 0679
- Anciennes 0289
- Linguistique 0290
- Modernes 0291

Littérature
- Généralités 0401
- Anciennes 0294
- Comparée 0295
- Médiévale 0297
- Moderne 0298
- Africaine 0316
- Américaine 0591
- Anglaise 0593
- Asiatique 0305
- Canadienne (Anglaise) 0352
- Canadienne (Française) 0355
- Germanique 0311
- Latino-américaine 0312
- Moyen-orientale 0315
- Romane 0313
- Slave et est-européenne 0314

PHILOSOPHIE, RELIGION ET THÉOLOGIE

Philosophie 0422
Religion
- Généralités 0318
- Clergé 0319
- Études bibliques 0321
- Histoire des religions 0320
- Philosophie de la religion 0322

Théologie 0469

SCIENCES SOCIALES

Anthropologie
- Archéologie 0324
- Culturelle 0326
- Physique 0327

Droit 0398
Économie
- Généralités 0501
- Commerce-Affaires 0505
- Économie agricole 0503
- Économie du travail 0510
- Finances 0508
- Histoire 0509
- Théorie 0511

Études américaines 0323
Études canadiennes 0385
Études féministes 0453
Folklore 0358
Géographie 0366
Gérontologie 0351
Gestion des affaires
- Généralités 0310
- Administration 0454
- Banques 0770
- Comptabilité 0272
- Marketing 0338

Histoire
- Histoire générale 0578
- Ancienne 0579
- Médiévale 0581
- Moderne 0582
- Histoire des noirs 0328
- Africaine 0331
- Canadienne 0334
- États-Unis 0337
- Européenne 0335
- Moyen-orientale 0333
- Latino-américaine 0336
- Asie, Australie et Océanie 0332

Histoire des sciences 0585
Loisirs 0814
Planification urbaine et régionale 0999
Science politique
- Généralités 0615
- Administration publique 0617
- Droit et relations internationales 0616

Sociologie
- Généralités 0626
- Aide et bien-âtre social 0630
- Criminologie et établissements pénitentiaires 0627
- Démographie 0938
- Études de l' individu et de la famille 0628
- Études des relations interethniques et des relations raciales 0631
- Structure et développement social 0700
- Théorie et méthodes. 0344
- Travail et relations industrielles 0629

Transports 0709
Travail social 0452

SCIENCES ET INGÉNIERIE

SCIENCES BIOLOGIQUES

Agriculture
- Généralités 0473
- Agronomie. 0285
- Alimentation et technologie alimentaire 0359
- Culture 0479
- Élevage et alimentation 0475
- Exploitation des pâturages 0777
- Pathologie animale 0476
- Pathologie végétale 0480
- Physiologie végétale 0817
- Sylviculture et faune 0478
- Technologie du bois 0746

Biologie
- Généralités 0306
- Anatomie 0287
- Biologie (Statistiques) 0308
- Biologie moléculaire 0307
- Botanique 0309
- Cellule 0379
- Écologie 0329
- Entomologie 0353
- Génétique 0369
- Limnologie 0793
- Microbiologie 0410
- Neurologie 0317
- Océanographie 0416
- Physiologie 0433
- Radiation 0821
- Science vétérinaire 0778
- Zoologie 0472

Biophysique
- Généralités 0786
- Medicale 0760

SCIENCES DE LA TERRE

Biogéochimie 0425
Géochimie 0996
Géodésie 0370
Géographie physique 0368
Géologie 0372
Géophysique 0373
Hydrologie 0388
Minéralogie 0411
Océanographie physique 0415
Paléobotanique 0345
Paléoécologie 0426
Paléontologie 0418
Paléozoologie 0985
Palynologie 0427

SCIENCES DE LA SANTÉ ET DE L'ENVIRONNEMENT

Économie domestique 0386
Sciences de l'environnement 0768
Sciences de la santé
- Généralités 0566
- Administration des hipitaux .. 0769
- Alimentation et nutrition 0570
- Audiologie 0300
- Chimiothérapie 0992
- Dentisterie 0567
- Développement humain 0758
- Enseignement 0350
- Immunologie 0982
- Loisirs 0575
- Médecine du travail et thérapie 0354
- Médecine et chirurgie 0564
- Obstétrique et gynécologie ... 0380
- Ophtalmologie 0381
- Orthophonie 0460
- Pathologie 0571
- Pharmacie 0572
- Pharmacologie 0419
- Physiothérapie 0382
- Radiologie 0574
- Santé mentale 0347
- Santé publique 0573
- Soins infirmiers 0569
- Toxicologie 0383

SCIENCES PHYSIQUES

Sciences Pures

Chimie
- Généralités 0485
- Biochimie 487
- Chimie agricole 0749
- Chimie analytique 0486
- Chimie minérale 0488
- Chimie nucléaire 0738
- Chimie organique 0490
- Chimie pharmaceutique 0491
- Physique 0494
- PolymÇres 0495
- Radiation 0754

Mathématiques 0405
Physique
- Généralités 0605
- Acoustique 0986
- Astronomie et astrophysique 0606
- Électronique et électricité 0607
- Fluides et plasma 0759
- Météorologie 0608
- Optique 0752
- Particules (Physique nucléaire) 0798
- Physique atomique 0748
- Physique de l'état solide 0611
- Physique moléculaire 0609
- Physique nucléaire 0610
- Radiation 0756

Statistiques 0463

Sciences Appliqués Et Technologie

Informatique 0984
Ingénierie
- Généralités 0537
- Agricole 0539
- Automobile 0540
- Biomédicale 0541
- Chaleur et ther modynamique 0348
- Conditionnement (Emballage) 0549
- Génie aérospatial 0538
- Génie chimique 0542
- Génie civil 0543
- Génie électronique et électrique 0544
- Génie industriel 0546
- Génie mécanique 0548
- Génie nucléaire 0552
- Ingénierie des systâmes 0790
- Mécanique navale 0547
- Métallurgie 0743
- Science des matériaux 0794
- Technique du pétrole 0765
- Technique minière 0551
- Techniques sanitaires et municipales 0554
- Technologie hydraulique 0545

Mécanique appliquée 0346
Géotechnologie 0428
Matières plastiques (Technologie) 0795
Recherche opérationnelle 0796
Textiles et tissus (Technologie) 0794

PSYCHOLOGIE

Généralités 0621
Personnalité 0625
Psychobiologie 0349
Psychologie clinique 0622
Psychologie du comportement 0384
Psychologie du développement ... 0620
Psychologie expérimentale 0623
Psychologie industrielle 0624
Psychologie physiologique 0989
Psychologie sociale 0451
Psychométrie 0632

THE UNIVERSITY OF MANITOBA

FACUTY OF GRADUATE STUDIES

COPYRIGHT PERMISSION

LOSSLESS COMPRESSION OF VOICEBAND DATA SIGNALS

BY

KENT W. NG

A Thesis/Practicum 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

Abstract

Today's public switched telephone networks (PSTN) are being upgraded to use digital transmission and switching. The main disadvantage of digital transmission is the higher bandwidth required. To alleviate this problem, much research has been devoted to the compression of digitized voice signals. With the recent growth of the Internet and the proliferation of modems and fax machines, non-voice traffic on the PSTN has increased dramatically. One problem with voiceband data traffic over a digital PSTN is the seemingly inefficient use of bandwidth. Consider that a 28.8 kbit/s v.34 modem requires a 64 kbit/s PCM voice channel for transmission. Another problem is that voice compression techniques tend to have adverse effects on non-voice signals such as modems and faxes. Only recently has the area of voiceband data signal compression been explored.

Literature search has turned up a small amount of research in the area of effects of voice compression systems on voiceband data signals but very little research in the area of actual compression of voiceband data signals. The most comprehensive works seems to have been done at the University of Salt Lake City in the late 1980s. They examined the use of Vector Quantization for compression of voiceband data (modem) signals. They only examined modem signals up to 4800bit/s. Since the time of their research modem speeds have increased to 28.8kbit/s and above. There is also one commercial company that offers a system that will demodulate modem signals, transmit the digital information over the telephone network and remodulate it at the other end. Their current system is capable of demodulating a 14.4kbit/s modem signal, transmitting it over a 16kbit/s channel and remodulating it at the other end.

This thesis experimentally examines whether or not it is possible to significantly compress digitized modem signals that are transmitted over the public telephone network in order to save bandwidth. The use of lossless compression schemes such as dictionary (Ziv-Lempel) and statistical (arithmetic and Huffman) compression were examined. These coders are able to reduce the bandwidth required for digitized modem signals by 5% to 25%. However, lossless coders have a variable bit-rate output and would be difficult to integrate into the public telephone network since it uses fixed bit-rate channels.

The basic conclusion from this thesis is that any attempt to compress modem signals would require many resources and excessive time to implement. A better solution is to adopt a completely digital network end-to-end. This is already happening with the move to ISDN and other digital transmission standards. While further research in this area may be of some academic interest, it appears to have little practical value for industry.

Acknowledgments

I would like to thank Dr. Howard Card and Dr. Robert McLeod for their guidance and support; Len Dacombe and Yair Bourlas from TRLabs, and Al Pollard from MTS for their help on the research project; Jeremy Sewall from TRLabs Edmonton for the use of his modem dataset; SriGouri Kamarsu, Alex McIlraith, Martin Meier, Roger Ng and Tapas Shome from the University of Manitoba for their assistance.

I would also like to thank my family for their encouragement and support throughout my time at university.

I greatly appreciate the financial support provided by Telecommunications Research Laboratories and by NSERC.

Table of Contents

Chapter 1 : Introduction

1.1 Purpose

Today's public switched telephone networks (PSTN) are being upgraded to use digital transmission and switching. In some cases, as in North America, the PSTN is almost completely digital. The transmission of digital signals has many advantages over analog transmission. For example, digital transmission provides protection from channel noise and distortion through error correction and signal regeneration. Another advantage of digital transmission is the commonality of digital signals compared to analog signals. Transmission of analog audio, video, and modem signals require different types of analog circuitry. The conversion of these signals to digital form means that they can be transmitted with the same equipment. The main disadvantage of digital transmission is the higher bandwidth required. As an example, a pair of coaxial cables is capable of carrying 13,200 simultaneous analog telephone conversations but only 4096 simultaneous digital telephone conversations. The use of digital transmission is not restricted to the PSTN. Digital transmission is finding its way into almost every communication system, including cellular telephone systems, satellite broadcast systems, and military transmission systems.

To alleviate the problem of higher bandwidth, there has been much research recently into the compression of digitized analog signals. Most of this research has focused on voice traffic. With the proliferation of modems and faxes, today's PSTN is carrying an increasing amount of non-voice traffic. One problem with voiceband data traffic over a digital PSTN is the seemingly inefficient use of bandwidth. Consider that a 28,800 bit/s modem requires a 64 kbit/s Pulse Code Modulated (PCM) voice channel for transmission. Figure 1 shows the stages of the transmission of a modem signal over the PSTN.

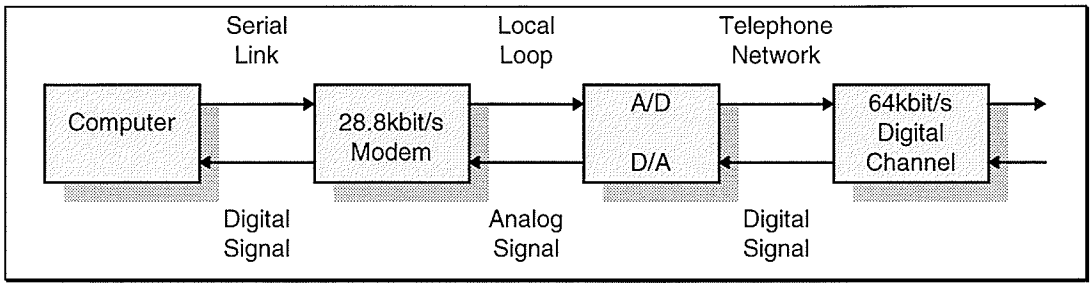

Figure 1

Until recently, there has been little research on the compression of voiceband data signals. The most comprehensive work of this kind appears to have been done by researchers at the University of Utah in Salt Lake City. They have produced three Ph.D. theses and several papers on the topic of vector quantization of voiceband data signals. There have also been many papers published on the effects on voice compression techniques on non-voice signals including voiceband data signals.

1.2 Scope

This thesis examines the use of lossless compression techniques on voiceband audio and modem signals transmitted over the PSTN. Experiments on the performance of standard lossless compression algorithms such as Ziv-Lempel dictionary compressors and statistical compressors on a dataset of digitized modem signals are conducted. An analysis of the performance of these algorithms will be used to determine the suitability of their use in the public telephone network.

Chapter 2 will briefly review analog versus digital signals and digital data compression. Highlights of the literature review of relevant research work pertaining to voiceband data compression are shown as well as commercial systems available.

Chapter 3 will briefly review the lossless compression algorithms used to compress the digitized modem signals. The two categories of lossless compressors, statistical and dictionary compressors are discussed. The variants used for the compression tests are also presented and discussed.

Chapter 4 presents the digitized modem signal dataset used for the compression tests. The results of the lossless compression algorithms on this dataset are discussed. The statistical information gathered on the dataset from the compression algorithms is also presented.

Chapter 5 will analyze the performance of the lossless compression algorithms and discuss the suitability of using these algorithms in the public telephone network.

Chapter 6 consists of recommendations and conclusions reached in this thesis.

Chapter 2 : Digital Communications and Compression

2.1 Digital Communications

Four categories can be defined for communications given both analog and digital information sources and transmission/storage methods being available:

1. Analog Data Signal - Analog Transmission/Storage
2. Analog Data Signal - Digital Transmission/Storage
3. Digital Data Signal - Analog Transmission/Storage
4. Digital Data Signal - Digital Transmission/Storage

Method 1 involves converting analog data signals to other forms of analog signals suitable for transmission or storage. This may require the use of modulation techniques such as amplitude modulation (AM) or frequency modulation (FM). These methods are commonly used for analog radio transmissions, audio cassettes, and vinyl records.

Method 2 involves converting analog signals to a digital form suitable for transmission or storage. The analog data signals are usually audio signals such as speech, music or video. Digital sampling and encoding are used to convert the analog signal into digital form. This can range for simple systems such as pulse code modulation (PCM) to more complex techniques such as delayed decision coding and transform coding.

Method 3 involves converting digital data signal into an analog form suitable for transmission or storage. This method is commonly used to store digital information onto computer disks or tapes and in communications between digital equipment over analog links such as the public telephone network, microwave, or satellite links. Digital signal modulation converts a digital bit stream into an analog signal suitable for transmission or storage. Digital signal demodulation performs the reverse operation of taking the analog signal and converting it back to a digital signal.

Method 4 involves converting digital information into a form suitable for digital transmission or storage. The digital signal used for transmission or storage is actually an analog signal that is directly proportional to the digital data signal. This is typically used for communications at the microchip and circuit board level. Examples include communications lines between microchips on a motherboard, or communications between two boards over a serial or parallel link. The transmission methods are fairly simple. For example, return-to-zero (RZ) encoding uses a +1 signal amplitude to represent a 1 bit and a 0 signal amplitude to represent a 0 bit. If necessary, slightly more

complicated methods can be used such as bipolar return-to-zero or Manchester coding. The main advantage to using digital signaling is that it is simple and has a low implementation cost. The main disadvantage is that it is not bandwidth efficient compared to method 3, typically allowing only a few bits to be transmitted per Hertz.

For the purposes of this thesis, we are only interested in method 2 and method 3. Method 3 is used by modems to allow computers to communicate over the public telephone network. Method 2 is used internally by the public telephone network to convert analog telephone signals using pulse code modulation to digital signals.

For more information on digital communications see: [OpSc89] [Strem90].

2.1.1 Analog Information to Digital Signals

In the 1950s, digital computers began to be used to simulate analog signal processing systems. This allowed designers to test and improve their analog systems before construction. As computer processing power increased, it became feasible to implement signal processing systems using digital computers. The advantage to using digital computers to work with analog signals was that common hardware could be used for signal transmission, storage and processing tasks. Different software algorithms would be run on the computer depending on the task.

In mathematical terms, an analog signal is a continuous signal in both time and amplitude while a digital signal is a discrete signal in both time and amplitude. Sampling is used to convert a continuous signal into a discrete time signal. The process of sampling can be described as follows:

$$x[n] = x_a(nT), \quad -\infty < n < \infty$$

$$x = \{x[n]\}, \quad -\infty < n < \infty$$

Given a continuous time fuction $x_a(t)$, a discrete time version of this signal is represented by a sequence of numbers x where the nth number is denoted by $x[n]$. Each $x[n]$ value is derived from the analog signal x_a sampled at time nT. T is the sampling period and n is the same n as the $x[n]$ sequence.

Under certain conditions both the continuous time and discrete time signals are equivalent. This condition is described by the sampling theorem. The sampling theorem states that the entire information content of a bandlimited continuous time signal can be recorded by sampling it at the rate F_s equal to twice the maximum frequency of the continuous time signal F_c. The sampling frequency F_s is also known as the Nyquist critical frequency. This is expressed mathematically as:

$$F_{sample} = 2F_{signal}$$

For simple analog to digital systems such as pulse code modulation (PCM), the digital sample is encoded using a linear (a-law) or logarithmic (μ-law) scale and is then ready for transmission or storage. This encoding process is called quantization. More complicated scheme such as delayed decision coders and transform coders use more sophisticated techniques to encode the digital samples. These advanced schemes are used to reduce the bandwidth and storage requirements required by the digital signal. For example a 4kHz bandlimited analog audio signal sampled at 8-bits 8kHz using PCM generates a bit stream at 64kbit/s. This bit stream would require an analog communications bandwidth of 64kHz for transmission using a simple digital binary signaling system. Using a delayed decision coder (such as vector quantization) to encode the digital bit stream, the bit rate could be reduced from 64 kbit/s to 16kbit/s. This would then require a communications bandwidth of 16kHz to transmit.

For more information on digital signal processing see: [OpSc89] [JaNo84].

2.1.2 Digital Information to Analog Signals

Modulation is commonly used to transmit information over long distances. It involves encoding an input electrical signal (which could be analog or digital) on to a carrier wave suitable for transmission or storage. Modulation was first used in radio transmissions for transmitting audio signals such as speech and music. Later, modulation was used in telephone networks to carry multiple telephone conversations on a single telephone line using frequency division multiplexing. With the advent of digital computers, modulation was used to transmit digital information over analog transmission facilities.

Modem stands for Modulator/Demodulator. The word "modem" is commonly used to describe a device used to transmit and receive digital information over analog communications facilities such as the public telephone network, satellite or microwave links. In its simplest form a modem takes a binary bit stream and uses it to modulate a carrier wave that is suitable for transmission over the analog link. It is also capable of the reverse operation of taking a modulated carrier wave received over an analog link and extracting a digital bit stream from it.

A sine-wave carrier is a common signal used for modulation. Three parameters can be modulated: amplitude, frequency, and phase. These three modulation methods are referred to as amplitude modulation, frequency modulation, and phase modulation, respectively. When these modulation methods are used for digital signals they are sometimes referred to as amplitude-shift keying (ASK), frequency-shift keying (FSK), and phase-shift keying (PSK).

The first modems used over the telephone network used a simple FSK modulation scheme to transmit a digital stream at 50 to 300 bits/s. More sophisticated PSK modulation schemes were then developed to transmit digital streams up to 4800 bit/s. Today's modem used a combination of a modulation methods called quadrature

amplitude modulation (QAM) or amplitude-phase keying (APK) to transmit digital streams at up to 33,600 bits/s.

A general sinusoidal-wave carrier modulation scheme can be described mathematically as:

$$\phi(t) = a(t)\cos(\theta(t)) \quad \text{where } \theta(t) = \omega_c t + \gamma(t)$$

In amplitude modulation the amplitude of the carrier $a(t)$ is varied with time in proportion to the input signal. Phase and frequency modulation are very closely related. In phase modulation the phase angle, $\gamma(t)$, is varied with the input signal. In frequency modulation the instantaneous frequency $\omega_i = d\theta/dt$ is modulated with the input signal.

ASK systems are the simplest to construct. When on-off keying is used, there is no power transmitted when there is no data to transmit. The main disadvantage of ASK is that for a given bit error probability, ASK requires the most power of any digital modulation scheme. FSK systems are generally more complex than ASK. FSK requires less power than ASK for a given bit error probability but is worse than PSK. PSK systems are generally more complex than FSK. PSK offers the best bit error probability with the lowest required power.

Variations on these 3 basic modulation schemes include:

- Coherent (synchronous) vs. Noncoherent detection
- Differential vs. Nondifferential coding
- M-ary (multi-level) modulation

Each of the various modulation schemes have advantages and disadvantages in terms of efficient use of bandwidth and performance in the presence of distortion. FSK, BPSK, DPSK, OQPSK, and MSK perform well in the presence of saturation-type nonlinearities. OOK and BPSK perform well with delay distortion. FSK performs well with quadratic-type distortion. OOK and coherent biorthogonal systems perform well in the presence of fading. In terms of bandwidth efficiency APK/QAM and M-ary PSK systems offer the best bits per Hertz performance.

The modem signals that we are primarily interested in are the APK/QAM type signals. These modulation schemes are the most commonly used for modems operating over the public switched telephone network.

For more information on modems see: [Mart90] [Strem90].

2.2 Digital Data Compression

Digital transmission and storage generally requires more bandwidth than the analog equivalent. In an attempt to solve this problem, a branch of mathematics called Information Theory was developed in the late 1940s by Claude Shannon working at Bell Labs. This branch of mathematics deals with storing and communicating messages. Digital data compression is derived from this since it is concerned with reducing the amount of redundancy in a message. Redundant information in a message requires extra bits to encode. If the redundant information can be eliminated then the size of the message can be reduced.

Digital data compression works by taking a stream of input symbols and transforming them into a set of codes based on a model of the input symbols. The model is a set of rules and data used to assign codes to the input symbols. Data compression techniques can be broken down into two general categories:

- Lossless Compression
- Lossy Compression

Lossless compression attempts to uses a model of the input symbols to encode them using fewer bits. The advantage of lossless compression is that the process is reversible. The entire input symbol set can be recovered from the codes emitted by the lossless compressor. The disadvantage of lossless compression is that it only provides a modest reduction in bandwidth, typically from 2:1 to 10:1. Lossless compression is useful for computer text files, object/executable files, and data files where the loss of any information through the compression process would be unacceptable.

Lossy compression also uses a model of the input symbols to encode them using fewer bits. In addition to this lossy compression uses the model to determine what information in the input symbol stream can be thrown away. The advantage of lossy compression is that it can provide a tremendous reduction in the bandwidth, typically from 10:1 to 100:1. The disadvantage of lossy compression is that the process is irreversible. The output of a lossy compressor can be used to reconstruct an approximation of the input symbols but is unable to reproduce the input symbols exactly. Lossy compression is usually used for digitized analog signals such as voice and video.

2.3 Literature Review

Human speech is well approximated by a signal which is bandlimited to frequencies below 4000Hz. Since the public telephone network was designed primarily to carry speech, its transmission channels are bandlimited to these frequencies to save bandwidth. The term "voiceband" will be used throughout this thesis to denote those audio signals that have been bandlimited for transmission over the telephone network.

This section will summarize past research in the area of voiceband data compression and resulting signal performance. Most of this research has concentrated on the use of lossy compression.

2.3.1 Differential Pulse Code Modulation

One of the first papers published in the area of voiceband data compression was "Differential PCM for Speech and Data Signals" by J. B. O'Neal and R. W. Stroh in 1972 [ONSt72]. This paper focused on how differential PCM systems, designed primarily for speech, performed with differential-phase modulation, single sideband and partial response data signals operating between 1200 and 9600 bit/s.

Input Signal → DPCM Optimized for signal ↓	Speech	Raised Cosine	Sine Spectrum (Partial Response)	2400bps Quad. Phase Modulation	9600bps partial response
Speech	11.41	-3.44	-3.18	-2.82	-7.00
Raised Cosine	-1.33	4.41	3.22	4.35	-0.54
Sine Spectrum (Partial Response)	0.32	4.25	3.39	4.33	-0.67
2400bps Quad. Phase Modulation	-0.87	4.33	3.29	4.44	-0.73
9600bps partial response	-0.70	2.19	1.72	2.00	0.65

Table 1

Input Signal → Compromise Predictor ↓	Speech	Raised Cosine	Sine Spectrum (Partial Response)	2400bps Quad. Phase Modulation	9600bps partial response
Case 1	5.06	2.78	2.39	3.08	-2.42
Case 2	3.50	3.51	2.92	3.65	-1.77
Case 3	7.80	0.79	0.80	1.33	-3.64
Case 4	8.75	0.02	0.12	0.60	-4.10

Table 2

Table 1 summarizes their basic results. The numbers in the table indicate the signal-to-noise ratio improvement (SNI) in dB of a DPCM system over a PCM system. The table shows the SNI for a 3-tap DPCM system optimized for certain signal types when other signal types are applied. All the input signals were sampled at 8kHz, 8bits.

Table 1 shows that a DPCM system optimized for a particular input signal will generally improve the SNI when that particular signal is applied. For example, a speech optimized DPCM system gives an 11.41dB improvement over PCM and a 2400bit/s optimized DPCM system gives a 4.44dB improvement over PCM. Negative numbers indicate a degradation in performance. For example, the speech optimized DPCM system has worse performance for all input signals other than speech.

O'Neal and Stroh go on to design four different compromise 3-tap DPCM systems for both speech and data signals. These results appear in Table 2. Case 1 involves designing a DPCM system optimized for 50% speech input and 50% data inputs. The case 2 DPCM system attempts to equalize the SNI factor for all signals. Case 3 attempts to make the SNI of both speech and data signals less than the maximum SNI possible for either. Case 4 improves the SNI of speech while keeping the SNI for data signals at an acceptable minimum value. One fact made clear by this study was that 3-tap DPCM systems offer little advantage over PCM for 9600bit/s partial response data signals even if a DPCM system was designed for that particular signal.

In 1990 the ITU published recommendation G.726 - "40, 32, 24, 16 kbits/s Adaptive Differential Pulse Code Modulation (ADPCM)". [ITU90] This recommendation states that 32kbit/s ADPCM links should give acceptable performance to modems not exceeding 2400 bit/s and should also accommodate 4800 bit/s signals with additional degradation over 64 kbit/s PCM. The recommendation also defines that 40 kbit/s ADPCM should give acceptable performance for modem signals not exceeding 12 kbit/s.

2.3.2 Delta Modulation of Data Signals

An important class of DPCM systems is delta modulation systems, also known as 1-bit or 2-level DPCM. Two studies that discussed the performance of voiceband data signals over delta modulated channels were O'Neal in 1974 [ONea74] and May, Zarcone, and Ozone in 1975 [MZO75]. Both papers came to similar conclusions regarding the performance of data signals over delta modulated channels. O'Neal examined the performance of five different modulation schemes: 1200 and 2400 bit/s phase modulation (PM), 4800 and 9600 partial response (PR), and 4800 single sideband (SSB). The SSB and PM signals employed raised cosine spectra while the PR signals employed sine spectra. The delta modulation system used had single integration with a perfect integrator in the feedback loop. The results of this paper indicated that a delta modulation channel equivalent to 16 kbit/s could not successfully transmit any of the test signals. A 32 kbit/s equivalent channel was capable of transmitting the 1200 bit/s PM signal successfully and was marginal for the 2400 bit/s PM signal. A 96 kbit/s equivalent channel was capable of transmitting all test signals successfully. A 68 kbit/s equivalent channel was required for meeting the noise requirements for C-type conditioned lines.

The results for the May, Zarcone, and Ozone paper were similar. There were five test signals used: 1200 bit/s frequency modulation (FM), 2400 bit/s phase modulation (PM), 4800 bit/s amplitude-phase modulation (APM), 4800 bit/s PM, and 9600 bit/s amplitude

modulation with vestigial side band (AMVSB). They found that the performance of a 64 kbit/s delta modulation channel was better than or at least equivalent to a 64 kbit/s PCM channel when both were subjected to transmission errors.

2.3.3 Low Delay Code-Excited Linear Prediction

In 1993, S. Dimolitsas and F. L. Corcoran published work on non-voice performance of the 16 kbit/s Low Delay Code-Excited Linear Prediction (LD-CELP) algorithm [DiCo93], [Dimo93]. This study examined the effects of the ITU standard 16 kbit/s LD-CELP algorithm for speech compression (Recommendation G.728) on non-voice signals including voiceband data signals.

The voiceband data signals examined in this paper were v.21 (300bit/s), v.22 (1200 bit/s), v.22bis (2400 bit/s), v.23 (1200 bit/s), and v.27ter (2400 bit/s). The 16 kbit/s LD-CELP algorithm can be operated in two modes: *voice* and *data*. The voice mode is used for both voice and in-band network signals while the data mode is only used for voiceband data signals. The results of the paper were obtained using the LD-CELP data mode, and show that 16 kbit/s LD-CELP offers acceptable performance for voiceband data signals at 2400 bit/s or lower.

2.3.4 Vector Quantization

The University of Utah Department of Electrical Engineering did a considerable amount of work in the area of data rate compression of voiceband data signals in the late 1980s. Several Ph.D. theses and papers were produced between 1985 and 1990 on this subject. The researchers at the University of Utah concentrated on developing algorithms for the vector quantization of voiceband data signals.

In 1985, D. O. Anderton produced his Ph.D. thesis "Vector Quantization of Voiceband Data Signals" [Ande85]. In his research, he developed what was called the adaptive baseband codebook vector quantizer (ABCVQ) for voiceband data signals. The test signals examined were: continuous phase frequency shift keying (CFSK), frequency shift keying (FSK), coherent binary phase shift keying (CBPSK), differential binary phase shift keying (DBPSK), differential quad phase shift keying (DQPSK), and differential octal phase shift keying (DOPSK). These signals ranged in data rate from 300 to 4800 bit/s.

The ABCVQ employed four parallel baseband template codebooks and one phase codebook to encode its input signal (See Figure 2). Codebook 1 applied to CFSK and CBPSK, codebook 2 to DBPSK, codebook 3 to DQPSK, and codebook 4 to DQPSK. The ABCVQ used a product search algorithm to select the template/phase vectors that best represented the input signal, converted it to passband and used this to encode the input signal. The centre frequency and power variance estimates were also encoded. The ABCVQ decoder simply used the frequency and power estimates to reconstruct the

passband codebooks and selected the encoded vector transmitted by the encoder. The ABCVQ successfully encoded all the test signals originally at 64kbit/s (sampled at 8-bits, 8kHz) with an output data rate of 16.172 kbit/s.

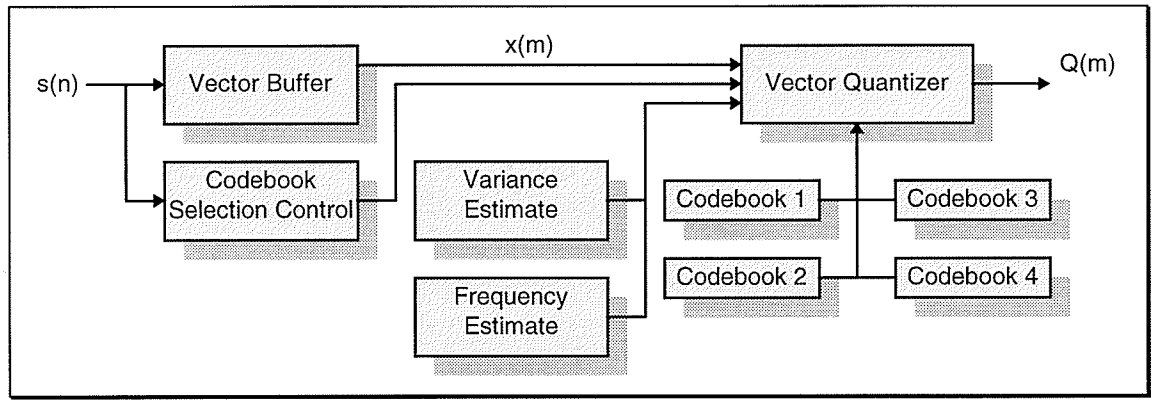

Figure 2

In D. S. Richards' 1988 Ph.D. thesis, "Baseband Template-Phase Vector Quantization of Voiceband Data Signals" [Rich88], he improves upon the ABCVQ by examining baud transition patterns, multi-stage vector quantization and finite-state vector quantization. Using the same test signals as Anderton, Richards developed a finite-state baseband template-phase vector quantizer (FS-BTPVQ) that successfully encoded the test signals with a total data rate between 10.028 to 13.216 kbit/s. The main disadvantage to the FS-BTPVQ was that it was sensitive to noise in the input signal.

T. D. Tran took a different approach in his 1987 Ph.D. thesis entitled "Data Compression of Bauded Signals using the Baseband Residual Vector Quantization Algorithm" [Tran87]. The baseband residual vector quantizer (BRVQ) converted a passband input signal to baseband and vector quantized the magnitude and phase residuals. Using the same test signals as Anderton, the BRVQ system successfully encoded the test signals with a total data rate between 11.3 to 16.5 kbit/s. The main advantage of the BRVQ algorithm over ABCVQ was that it achieved comparable performance with one third the complexity.

2.3.5 Commercial Compression Systems

DSP Software Engineering of Bedford, MA has developed a system called InvsiLink that is designed to demodulate modem/fax signals, to transmit them digitally and then to re-modulate them at the destination. Figure 3 shows a block diagram of where the InvsiLink system would fit into the telephone network. The system currently supports G3 fax signals and v.22bis/v.32bis modem signals. These signals are demodulated and transmitted over a 16 kbit/s channel. Four of these signals can be multiplexed onto a 64 kbit/s voice channel (DS0).

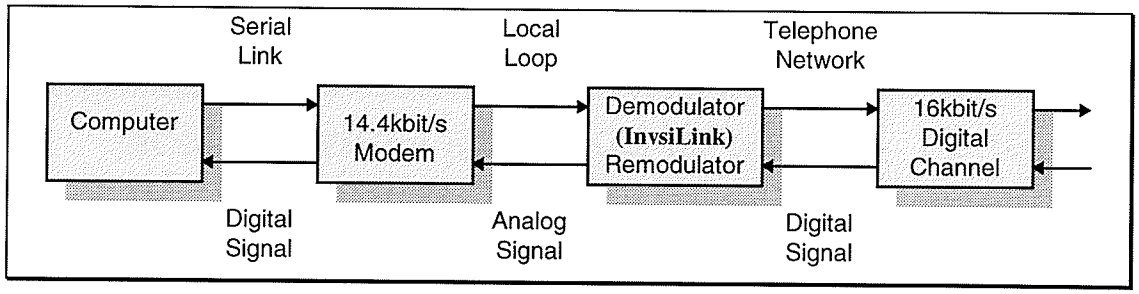

Figure 3

Lin, Kurtz, and McCarthy [LKM90] developed a low-rate encoder/decoder to transmit voiceband modem signals over radio links. The system used a TMS32020 DSP chip and had a total data rate of 14.5 kbit/s. It successfully transmitted Bell 103, Bell 212A, v.21, v.22, v.22bis, and v.26ter modem signals 99.8% error free with an RF link error rate below 10^{-7}.

2.3.6 Lossless Compression

There has been some research in the area of lossless compression of audio signals but little if any in the area of lossless compression of voiceband data signals. The main reason there has been little work in this area is due to the fact that lossy compression systems are able to take advantage of audio signal parameters and characteristics of human hearing to achieve much higher compression ratios than lossless coders are able to achieve. Work in the area of lossless compressors has focused on using them as a second level of coders, for example, using Huffman coding on an ADPCM or vector quantized bit stream. A more sophisticated lossless compression system using LZW with a LRU/LFU maintained codebook and arithmetic coding is presented in [ShHe94] for encoding a 7 bit ADPCM bit stream.

2.3.7 High Speed Modem Performance

With the recent interest in the internet and the increased use of modems, several general US computer magazines studied the effects of digital voice compression systems on high speed modems. PC Magazine reported a performance decrease between 10% to 70% for 14.4 kbit/s v.32bis modems transmitting over 32 kbit/s ADPCM channels [Satc94]. In another study, Byte Magazine showed that v.34 modems were only able to connect at 28.8 kbit/s and only under ideal line conditions [Vaug95]. The best performance of a majority of v.34 modems tested was 26.8 kbit/s. The poor performance was attributed to trunk components and switches that artificially limited bandwidth that v.34 modems required to operate effectively rather than quality of copper in local loops. The Byte Magazine study also sited voice compression systems as a growing problem for high speed modems. Such systems could reduce modem performance to 9600 bps and lower.

Chapter 3 : Review of Lossless Compression

3.1 Lossless Compression

This chapter reviews the different type of lossless compression algorithms. These algorithms are applied in Chapter 4 to a dataset of digitized audio, modem and fax signals and the performance between the alogrithms compared.

As was mentioned in Chapter 2, digital data compression works by taking a stream of input symbols and transforming them into a set of codes based on a model of the input symbols. The model used to encode the input symbol stream is perhaps the most important part of any data compression system. The more accurate the model is in representing the input symbols the fewer bits can be used to encode the input symbols. Models can be either static (not changing with time) or adaptive (changing with time).

Static models have the advantages of being suited to a particular set of input symbols and not having to update the model over time. The disadvantage of static models are that the model must be built first by scanning the input symbols and the model must be transmitted or stored with the encoded input symbols or it must be known beforehand by both the compressor and decompressor. Another disadvantage of the static model is that it is only suited to one particular set of input symbols. If a different set of input symbols with different probabilities is used then the static model may not perform as well.

Adaptive models have the advantages of being able to adapt to changes in the probabilities of the input symbols and of not having to transmit or store the model. The disadvantage of adaptive models is that the model has to be constantly updated to reflect the statistics of the input symbols. This is only a minor disadvantage compared to the flexibility provided by adaptive models. For this reason most lossless compression research in the past has concentrated on using adaptive models. This thesis will only examine adaptive models.

Two categories of lossless compression algorithms were examined:

- Statistical Coders - Huffman and Arithmetic Coders
- Dictionary Coders - Ziv-Lempel Coders (LZ77 and LZ78)

Also examined were combination compressors that used dictionary coders that were further encoded using a statistical coder. All lossless compression algorithms examined here were adaptive. The C source code used for all the compression algorithms were

derived from [Nels92]. For more background information about lossless compression see [BCW90].

3.2 Statistical Compression

Statistical compression algorithms attempt to encode input symbols by modeling the probability of their occurrence in the input symbol stream. More frequently appearing input symbols are encoded using fewer bits. Less frequently appearing input symbols are encoded using more bits. Two statistical compression algorithms were exploed in this thesis:

- Adaptive Huffman compression
- Adaptive arithmetic compression

All the models used for the statistical compression algorithms presented in this thesis are order 0 or context free models. This means that the probability of any given input symbol is calculated based solely on the frequency of its appearance and does not take into account the input symbols that preceded it in the input symbol stream. Since the statistical algorithms use an order 0 model, the only way they can compress the input symbol stream is if the probabilities of the input symbols are not equal (non-uniform distribution).

3.2.1 Adaptive Huffman Algorithm

Huffman compression, developed in the 1950s, was one of the first and probably the best known compression algorithm [Huff52]. The original version of the Huffman algorithm used a static model. Various adaptive versions were developed in the 1970s. The basic ideas behind Huffman compression come from Shannon-Fano coding developed by C. Shannon at Bell Labs and R. M. Fano at M.I.T. in the late 1940s. Shannon-Fano coding used a model based on the probability of occurrence of every input symbol in a given message. Given these probabilities, a table of codes could be constructed with the following properties:

- Codes are variable length with an integral number of bits
- Codes for low probability symbols use more bits
- Codes for high probability symbols use fewer bits
- Codes have a unique prefix attribute such that they can be decoded correctly

Both Shannon-Fano and Huffman algorithms store the codes in binary trees. The difference between the two algorithms is the method used to build the tree. The Shannon-Fano tree is built from the top down while the Huffman tree is built from the bottom up. The two algorithms also differ in terms of the code size used to encode the input symbols. In general, Shannon-Fano and Huffman coding are very close in

performance but Huffman is always at least as efficient as Shannon-Fano and sometimes is more efficient in coding the input symbols. For this reason Huffman became one of the predominant methods of coding. Huffman also proved in his paper that there was no way to improve the efficiency of coding using any other integral bit width coding method.

Building the Huffman tree is a simple task. The following steps are used to build the tree:

1. Find two free nodes with the lowest weights from the list of free nodes.
2. Create a parent node with a weight equal to the sum of the two child nodes.
3. Add the parent node to the list of free nodes and remove the two child nodes.
4. Assign one of the child nodes as 0 and the other as 1.
5. Repeat 1 to 4 until there is only one free node left. This node becomes the root node.

Consider, for example, the following symbols and their occurrence frequency (weight): A (3), B (2), C (1). The Huffman tree for these symbols is shown in Figure 4.

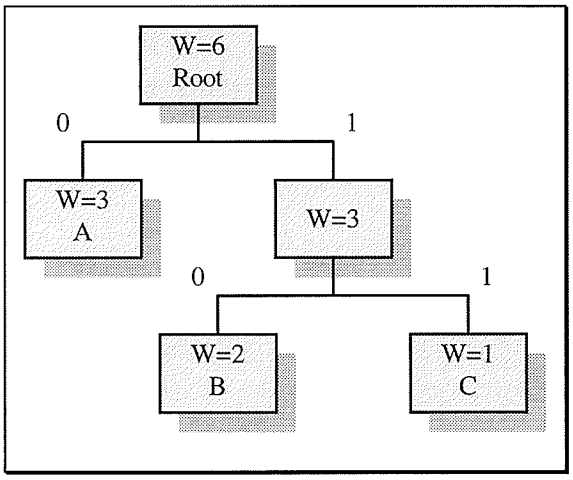

Figure 4

The symbols would be assigned the following codes: A = 0, B = 10, and C = 11.

Although the Huffman algorithm is not suited for use in adaptive coding, it can be converted to use an adaptive method. This is done by modifying the method in which the Huffman tree is built. Figure 5 and Figure 6 show block diagrams for the Huffman compression and decompression algorithms.

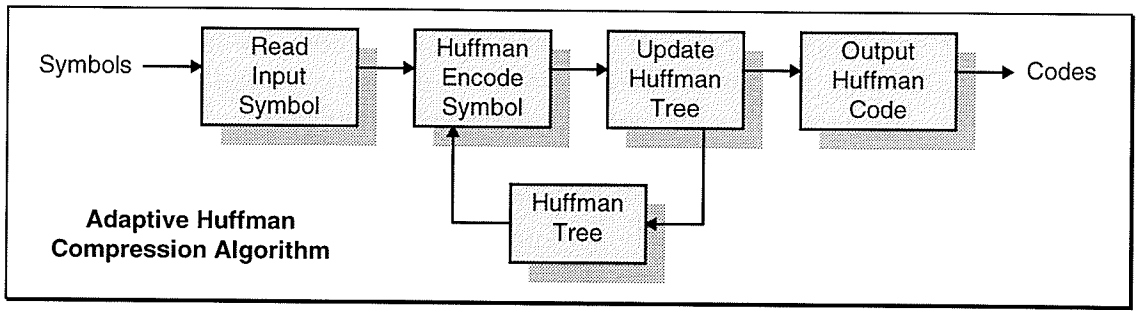

Figure 5

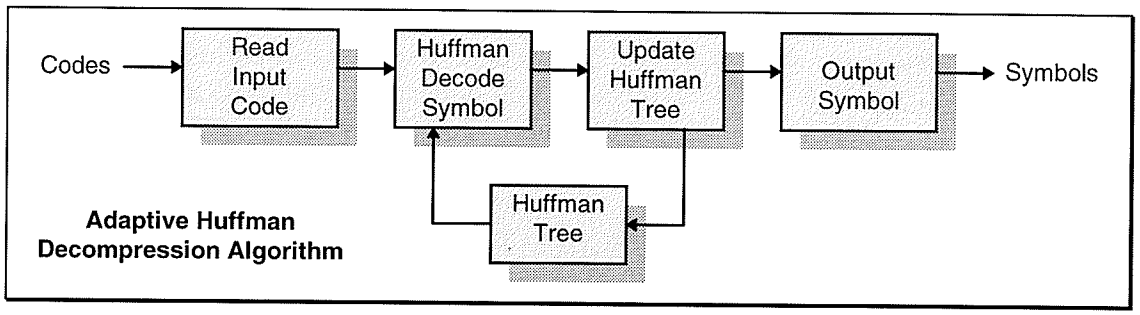

Figure 6

Both the compressor and decompressor start off with the same model. The compressor updates the model everytime an input symbol is encoded. The decompressor updates the model everytime a code is decoded. Both the compressor and decompressor will have the same model at any given input symbol/code.

The UpdateModel procedure performs two functions. The first is to increment the weight of the leaf node and all its parent nodes. The second is to ensure that the Huffman tree obeys the sibling property. A tree exhibits the sibling property if the nodes can be listed in order of increasing weight and if every node appears adjacent to its sibling in the list. This basically involves checking each node as its weight is updated. If the weight is larger than the weight of a higher level node then the nodes are swapped.

Two other enhancements are used in the adaptive Huffman algorithm used for testing. The first is rescaling the tree and the second is using escape codes.

The adaptive Huffman algorithm used assumes that integers are 16 bits long. The tree is rescaled at regular intervals by dividing all the weights in half and rebuilding the tree. This prevents any overflow of the 16 bit integers. A side effect of the rescaling process is that it may help improve the compression efficiency. This is due to a decay effect, in which the statistics of the current input symbols are a better measure of the probability than older input symbols.

Rather than initializing the Huffman tree with all 256 (8 bit) input symbols equally probable, escape codes are used instead. The Huffman tree starts out as a nearly empty

tree with two symbols, the end of stream code and the escape code. When a new input symbol is encountered by the compressor, it emits the escape code and then emits the new input symbol. The new symbol is then added to the Huffman tree.

The algorithm used for this thesis was an unmodified version of the adaptive Huffman compression algorithm shown in [Nels92].

3.2.2 Adaptive Arithmetic Algorithm

Arithmetic compression was first developed in the late 1970s. It was similar in function to Huffman but had some properties that allowed it to achieve superior compression compared to Huffman. This was done by using non integral bit length code. The theory behind arithmetic compression uses a single floating point number to replace a stream of input symbols.

Consider the following message: BABCB. The symbol probabilities and their cumulative range between 0 and 1 are shown in Table 3.

Symbol	Probability	Range
A	1/5	0.0 >= range > 0.2
B	3/5	0.2 >= range > 0.8
C	1/5	0.8 >= range > 1.0
Total	1/1	1.0

Table 3

The C pseudo code listed below show the basic operations of the arithmetic compressor and decompressor using a static model

Basic Arithmetic Compressor:

```
low = 0.0;
high = 1.0;
while ( (c = GetChar(input)) != END_OF_FILE)
{
  range = high - low;
  high = low + range * HighRange( c );
  low = low + range * LowRange( c );
}
output( low );
```

Basic Arithmetic Decompressor:

```
number = InputCode();
for ( ; ; )
{
  symbol = FindSymbolBetweenRange( number );
  PutChar( symbol );
```

```
    range = HighRange( symbol ) - LowRange( symbol );
    number = number - LowRange( symbol );
    number = number / range;
}
```

The arithmetic compression algorithm would code this message as follows:

Symbol	Low	High	Range
Initialize	0.0	1.0	1.0
B	0.2	0.8	0.6
A	0.2	0.32	0.12
B	0.224	0.296	0.072
C	0.2816	0.296	0.0144
B	0.28448	0.29312	-
Output	0.28448		

Table 4

The final value of 0.28448 would be used as the code for the message BABCB.

Encoding and decoding a series of floating point numbers may seem to be a complicated task. In practice the floating point numbers can be approximated with 16 or 32 bit integers. The actual algorithm used for this thesis utilized 16 bit integers, adaptive modeling with no escapes codes, and arithmetic symbol table rescaling. All of the 256 input symbols are given an initial count of 1. Rescaling the arithmetic symbol table is done for the same reason as rescaling the Huffman tree: namely, preventing overflow. The number of bits used in the symbol table must be at least two less than the number of bits in the high and low registers and the bits in the symbol table; also the bits in the high or low registers must not exceed the bits used in arithmetic calculations during the coding process. To meet these two restrictions the high and low registers are stored in 16 bit integers and the arithmetic calculations are performed in 32 bit integers. The use of 16 bit integers for the high and low registers means the maximum value of the frequency count for any symbol in the symbol table is limited to 14 bits. That means the table must be rescaled before the total count reaches 16383.

Figure 7 and Figure 8 show block diagrams for the adaptive arithmetic compression and decompression algorithms.

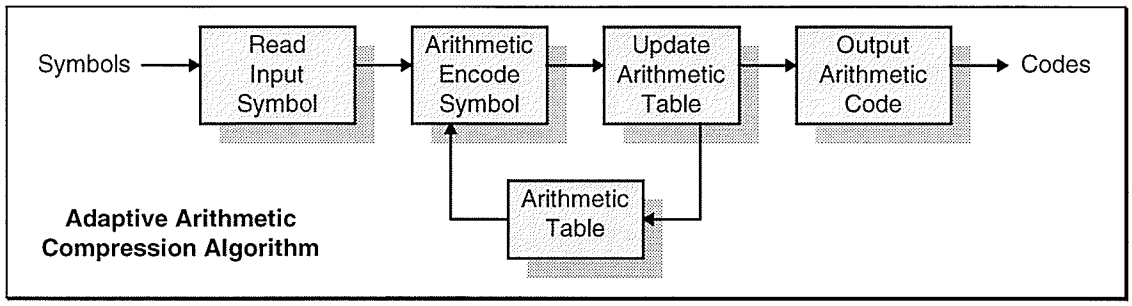

Figure 7

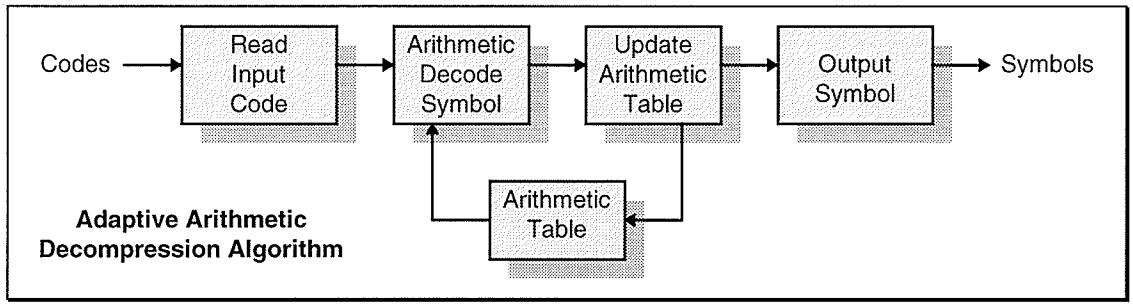

Figure 8

The algorithm used for this thesis was a modified version of the adaptive arithmetic compression algorithm shown in [Nels92]. The modifications involved changing the static order-0 arithmetic compressor into an adaptive order-0 arithmetic compressor. For a more detailed discussion on how the arithmetic compression algorithm works see [WNC87].

3.3 Dictionary Compression

Most modern dictionary compression techniques can be traced back to the work done by two Israeli researchers, Jacob Ziv and Abraham Lempel, in the late 1970s. Their work is described in two papers in the *IEEE Transactions on Information Theory*. "A Universal Algorithm for Sequential Data Compression" was published in 1977 and is known as LZ77. [ZiLe77] "Compression of Individual Sequences via Variable-Rate Coding" was published in 1978 and is known as LZ78. [ZiLe78] This section will discuss how the LZ77 and LZ78 algorithms work and which variants were used for testing our modem dataset.

3.3.1 LZ78 Algorithm

The output of LZ78 consists of a series of tokens that represent a given phrase and a single character. Both the encoder and decoder start with nearly empty dictionaries. As

each character is encoded or decoded, it is added to the dictionary. For any characters that the encoder hasn't seen before, it outputs an empty phrase following the character.

A practical implementation of LZ78 did not appear until 1984 when Terry Welch published "A Technique for High-Performance Data Compression" in *IEEE Computer*. [Welc84] The technique Welch used for the implementation of the Unix Compress program is referred as LZW compression. The major difference between LZW and LZ78 is that LZW only outputs phrases. This was accomplished by preloading the phrase dictionary with single symbol phrases equal to the total number of symbols in the alphabet.

Consider the following phrase: "**LZW LZ78 LZSS LZ77**". Using the LZW algorithm, the phrase would be encoded as shown in Table 5.

Input Symbol	Output Code	Phrase added to Dictionary
"LZ"	"L"	<256> = "LZ"
"W"	"Z"	<257> = "ZW"
" "	"W"	<258> = "W "
"L"	" "	<259> = " L"
"Z7"	<256>	<260> = "LZ7"
"8"	"7"	<261> = "78"
" "	"8"	<262> = "8 "
"LZ"	<259>	<263> = " LZ"
"S"	"Z"	<264> = "ZS"
"S"	"S"	<265> = "SS"
" "	"S"	<266> = "S "
"LZ7"	<263>	<267> = " LZ7"
"7"	"7"	<268> = "77"
<EOF>	"7"	

Table 5

Figure 9 and Figure 10 show the basic block diagrams for the LZW compressor and decompressor:

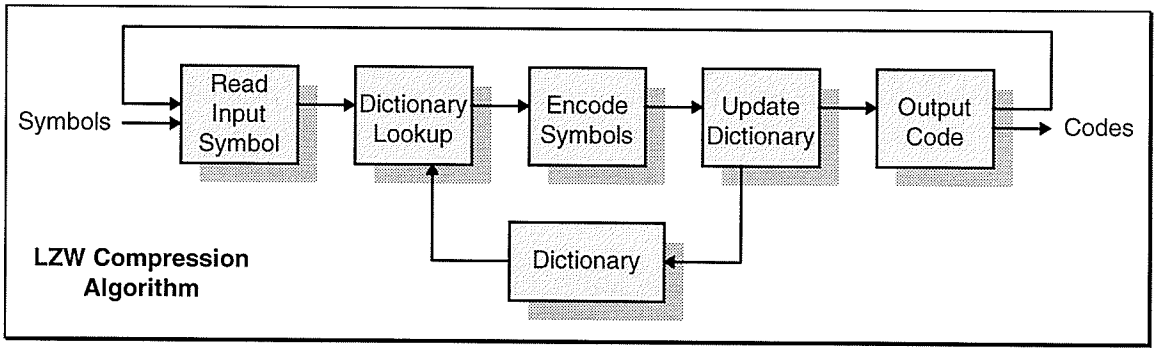

Figure 9

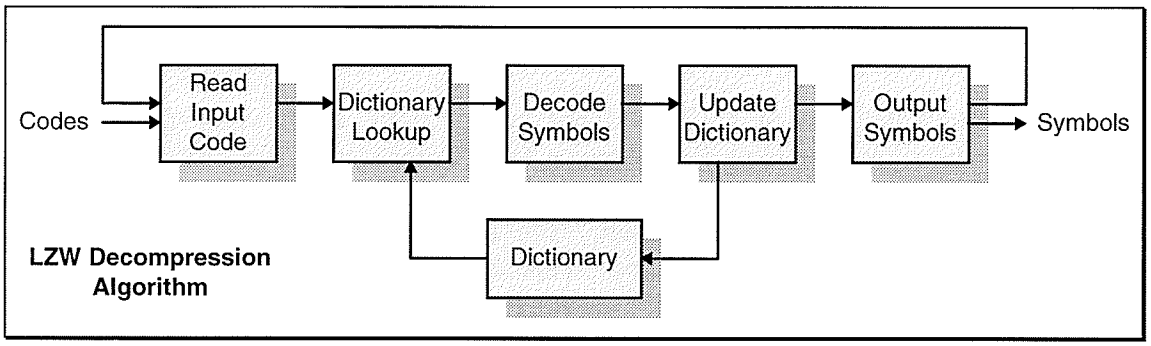

Figure 10

Two variants of the LZ78 algorithm, called LZW and LZWV, were used for testing the modem dataset. The LZW algorithm used a fixed dictionary size of 10 bits for 1024 codewords, 12 bits for 4096 codewords, and 14 bits for 16384 codewords. When the dictionary was filled it was discarded and a new dictionary built. These were called LZW-10, LZW-12, LZW-14 respectively. The LZWV algorithm used a variable sized codeword starting at 9 bits and expanding to a larger code size when needed. Three code sizes of 11 bits, 13 bits, and 15 bits were tested. These were called LZWV-11, LZWV-13, and LZWV-15 respectively. LZWV starts with a 9 bit code giving a codebook size of 512 codewords. The code size is expanded as needed to a maximum of 11 bits for 2048 codewords (LZWV-11), 13 bits for 8192 codewords (LZWV-13), and 15 bits for 32768 codewords (LZWV-15).

The LZW and LZWV algorithms used for this thesis were unmodified versions of the LZW algorithms shown in [Nels92].

3.3.2 LZ77 Algorithm

LZ77 works by maintaining a text window containing the most recently seen text. This window is divided into two parts. The first part is a large block of previously seen text. The second part is a small lookahead buffer of text read from the input stream and not yet encoded. LZ77 works by trying to match strings in the lookahead buffer with string in the large block of previously seen text. The output of LZ77 consists of a three part token. The first part contains the match position, the second part contains the match length and the third part contains the first character that follows the matched string. If a match is not found then the match position and length are set to zero in the token and the single character is emitted. The matched character or string is then moved into the large block of previously seen text and new characters are read in to replace the characters moved out.

Table 6 shows an example of what might be in the LZ77 text window and lookahead buffer.

	Previous Text Window	Look Ahead Buffer	
	11111111112222222233 01234567890123456789012345678901	01234567	
...	for(i=0; i<MAX-1; i++) for(j=i;	j<MAX-1;	j++);

Table 6

The LZ77 algorithm will attempt to match the string "**j<MAX-1;**" with anything in the previous text window. The only match is the symbol "**j**" at position 27 in the text window. The token emitted is (27, 1, "**<**"). This indicates a match at position 27 of the text window for a length of 1 symbol. The string "**j<**" is then shifted from the lookahead buffer into the previous text window and two new symbols are read into the lookahead buffer. The table below shows the results of the window and buffer shift.

	Previous Text Window	Look Ahead Buffer	
	11111111112222222233 01234567890123456789012345678901	01234567	
.fo	r(i=0; i<MAX-1; i++) for(j=i; j<	MAX-1; j	++); pr

Table 7

Next the "**MAX-1; j**" string will be matched with the previous text window. A match with "**MAX-1;** " will be found at position 9 of the previous text window for a length of 7 symbols. The token emitted is (9, 7, "**j**"). The window and buffer would then be shifted 8 symbols. The table below shows the results:

	Previous Text Window	Look Ahead Buffer	
	11111111112222222233 01234567890123456789012345678901	01234567	
; i	<MAX-1; i++) for(j=i; j< MAX-1; j	++); pri	ntf("%d

Table 8

Figure 11 and Figure 12 show the basic block diagrams for the LZ77 compressor and decompressor:

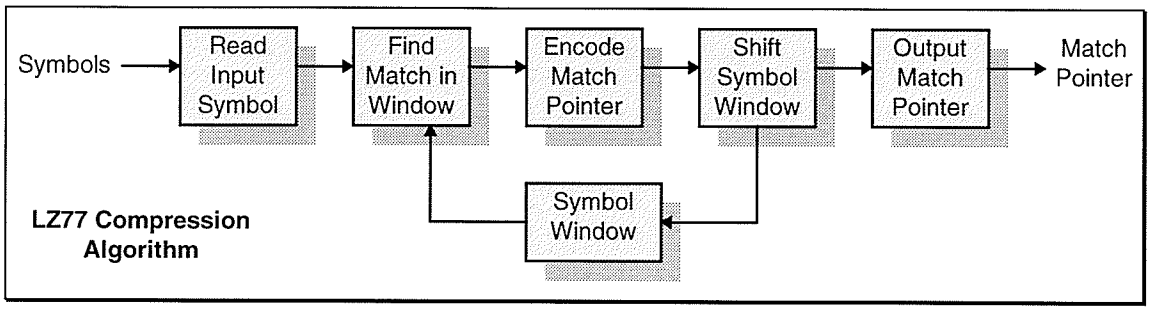

Figure 11

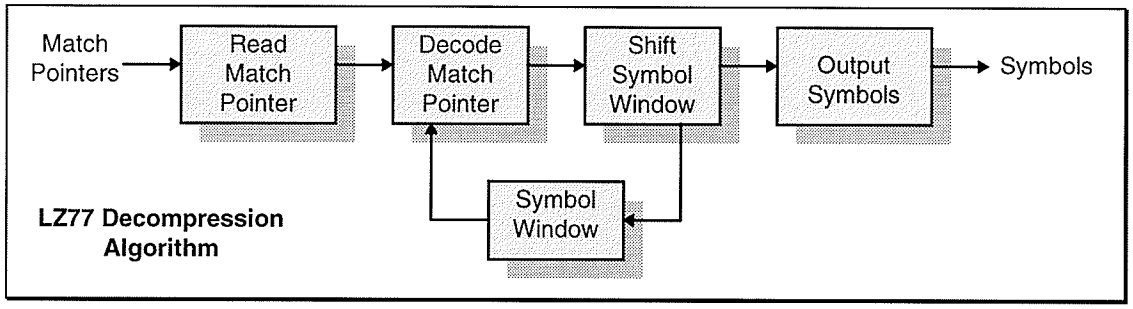

Figure 12

There are several major problems with LZ77 in its original form. The first is that a string comparison must be made with the look ahead buffer and every position in the text window. The second is the true use of a sliding window of text is inefficient since it would require shifting the text in the window everytime a match is made. The third problem is inefficiency when attempting to encode strings not found in the previous text window.

3.3.2.1 LZSS

LZSS seeks to improve upon LZ77 in three areas. The first improvement is to get rid of the character following every pointer. LZSS emits two different types of tokens. The first is a character and the second is a pointer. A single bit is used to distinguish between the two. This allows LZSS to freely mix characters and pointers. Characters are used for encoding when a pointer would take up more space. The second improvement is the use of a fixed buffer instead of an actual sliding window through the text. This eliminates the need to use expensive memory move operations. The third improvement is to add phrases moved out of the look ahead buffer into a binary tree to allow fast searches for matching phrases.

Figure 13 and Figure 14 show the basic block diagrams for the LZSS compressor and decompressor:

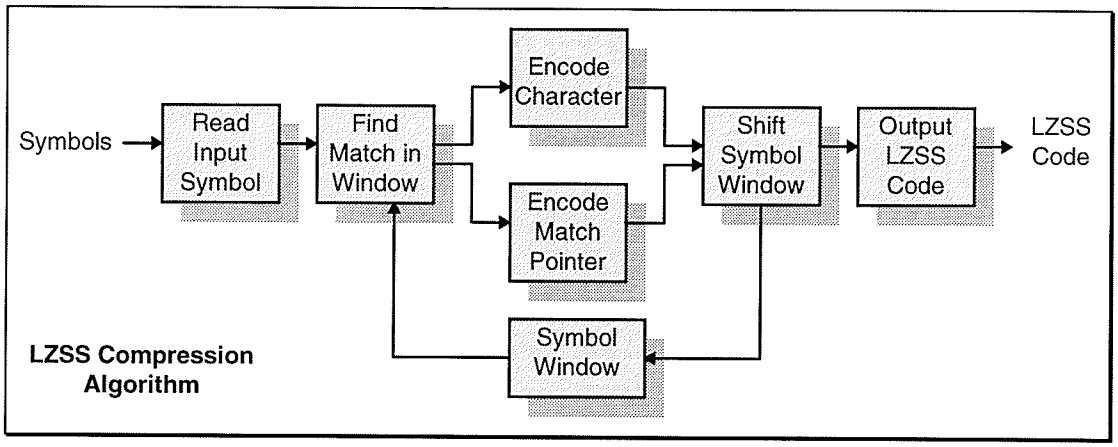

Figure 13

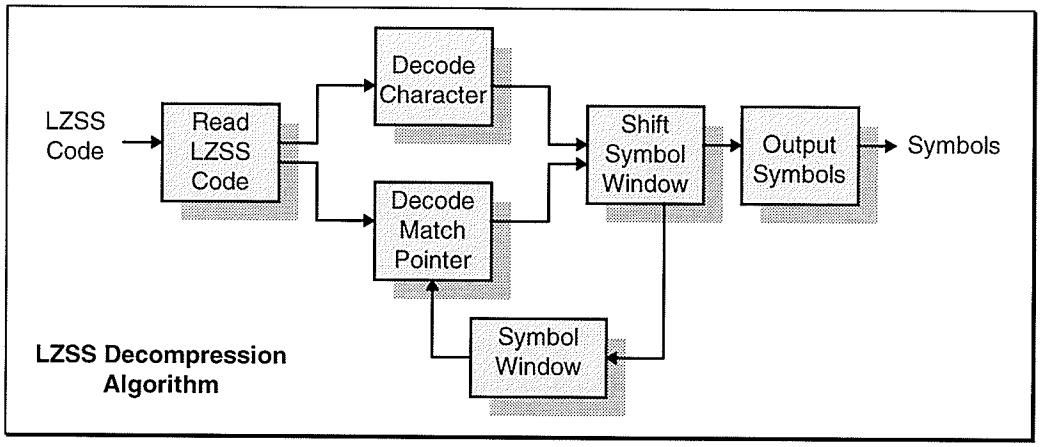

Figure 14

The LZSS algorithm used for this thesis was an unmodified version of the algorithm shown in [Nels92]. The LZSS algorithm was tested on the modem dataset using window sizes of 10 bits, 12 bits, and 14 bits, and match length sizes of 2 bits, 4 bits, and 6 bits.

3.3.2.2 LZH

The LZH algorithm is described in [Bren87]. The LZH algorithm used in this thesis was a minor modification of the LZSS algorithm presented in [Nels92]. A 12 bit window size and a 4 bit match length are used. The 4 bit match length is encoded using a static Huffman table shown in Table 9.

Match Length	Bits	Huffman Code
2	0000	0
3	0001	100
4	0010	101
5	0011	11000
6	0100	11001
7	0101	11010
8	0110	11011
9	0111	111000
10	1000	111001
11	1001	111010
12	1010	111011
13	1011	1111000
14	1100	1111001
15	1101	1111010
16	1110	1111011
17	1111	1111100

Table 9

3.3.2.3 LZFGH

The LZFGH algorithm is a combination of the LZH algorithm and the LZFG-A1 algorithm presented in [FiGr89]. The LZFGH algorithm used in this thesis is another modification of the LZSS algorithm presented in [Nels92]. LZSS uses a single bit to indicate whether a token is a symbol or pointer. This means that a single 8 bit symbol takes 9 bits to encode. This becomes wasteful if many symbols need to be encoded. The LZFG-A1 algorithm uses a 4 bit count to indicate the number of symbols to copy. A count of 0 indicates a pointer.

The LZFGH algorithm used for testing is a hybrid of the LZH and LZFG-A1 algorithm. It uses three output tokens shown in Table 10. The match pointer is the same one used in LZSS with the match length Huffman-encoded as in LZH. Strings of symbols are encoded using the symbol string token. It begins with an 8 bit count which indicates how many symbols are to follow. Strings shorter than 10 symbols are encoded as single symbols since the overhead bits are fewer.

Token	Prefix	Description
Single Symbol	0	8 bit symbol follows
Symbol String	10	8 bit count and followed by string of literals with length indicate by count
Match Pointer	11	12 bit window position pointer and Huffman coded 4 bit match length

Table 10

Figure 15 and Figure 16 show the basic block diagrams for the LZFGH compressor and decompressor:

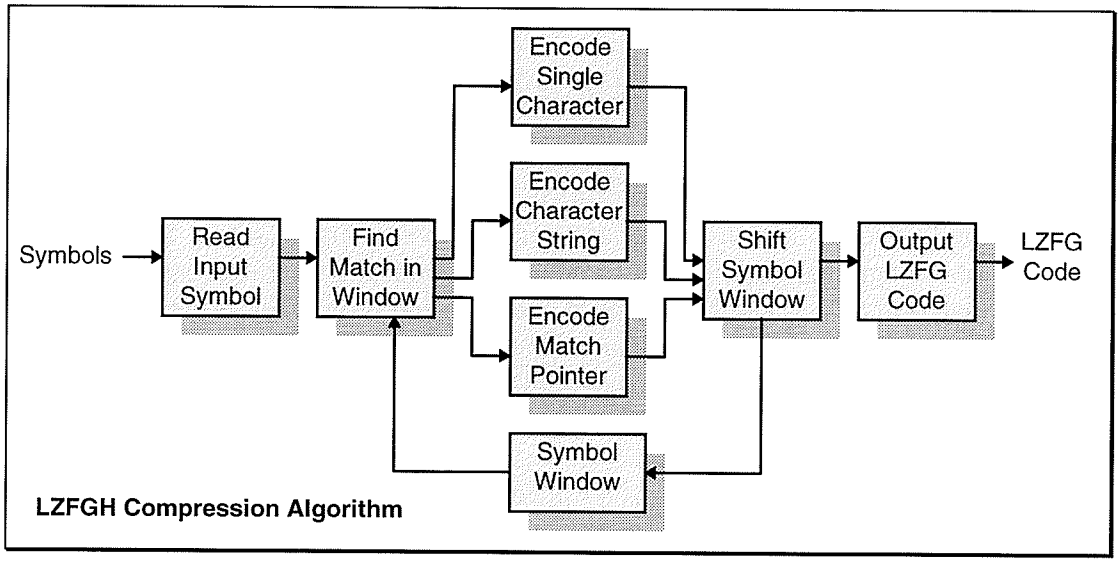

Figure 15

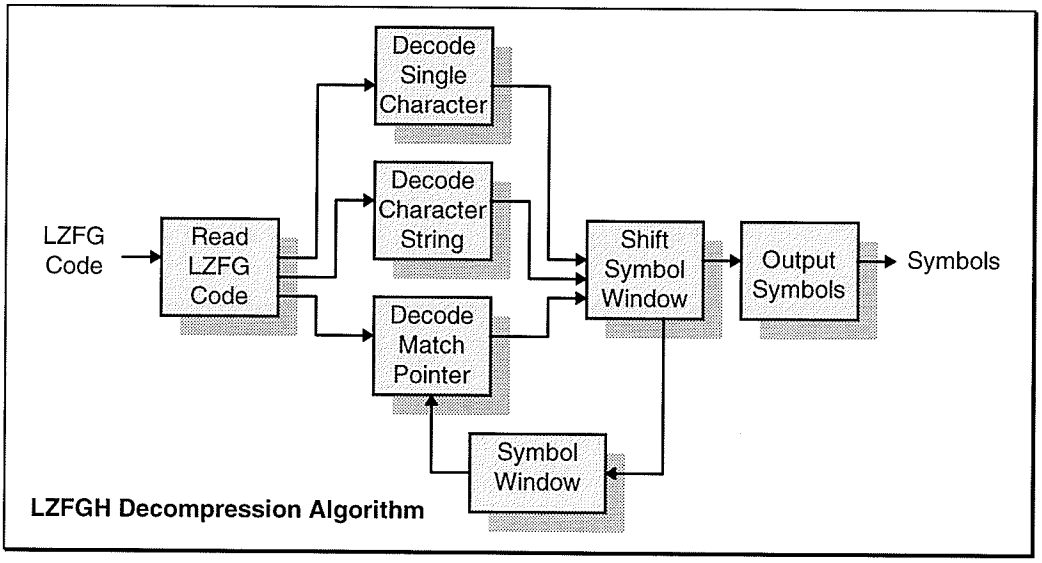

Figure 16

3.4 Combined Dictionary-Statistical Compression

These compression algorithms combined variants of the Ziv-Lempel compressors with a statistical compressor. Two combined dictionary-statistical compressors were tested on the dataset. The first one called LZAH combined a variant of the LZSS algorithm with

adaptive Huffman compression. The second called LZWAH combined the LZW algorithm with adaptive Huffman compression.

3.4.1 LZSS with Adaptive Huffman Coding (LZAH)

The LZAH algorithm is a modified version of the LZSS coder. The principal change is that literals are coded using the adaptive Huffman compressor. The first bit used for identification of literals or match pairs has been removed. In its place, the adaptive Huffman end-of-stream character is used to denote the end of a character string and the start of a LZ match pointer. The LZ match pointer is used only when it can encode the input symbols in fewer bits than the equivalent average Huffman bit length.

Figure 17 and Figure 18 show the basic block diagrams for the LZAH compressor and decompressor:

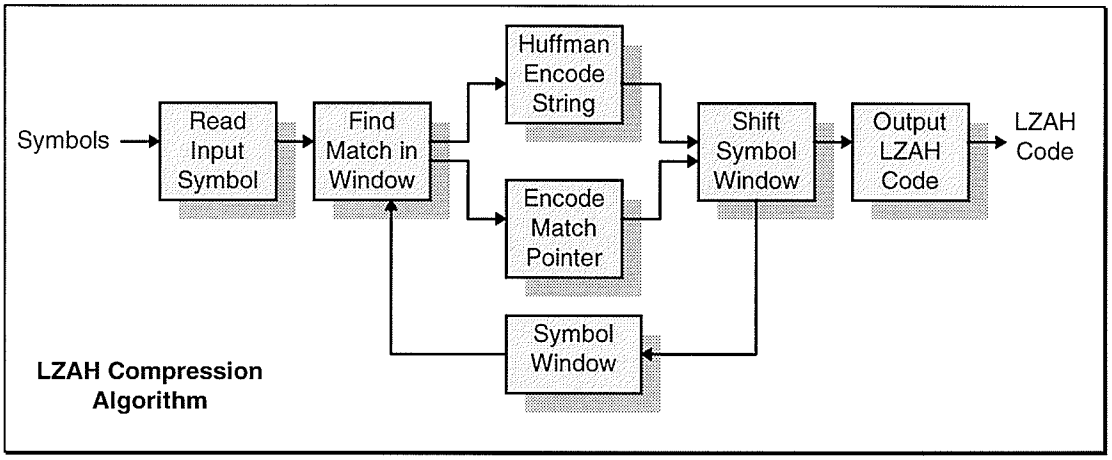

Figure 17

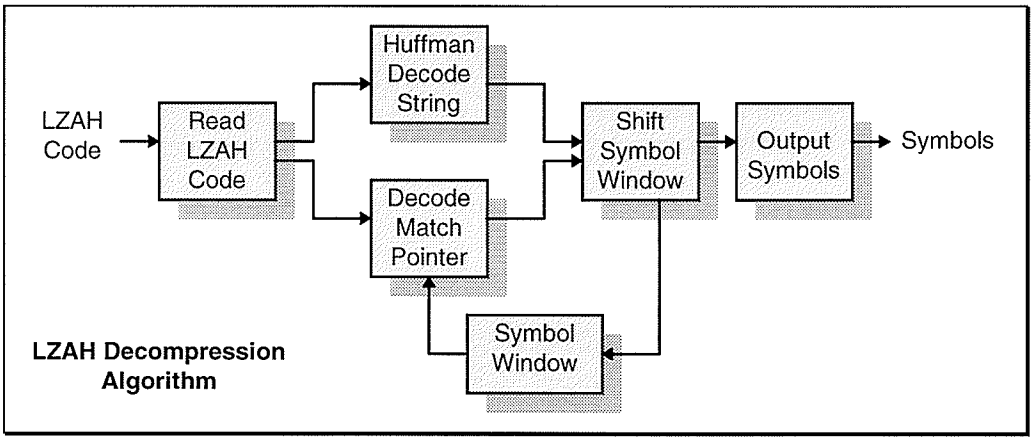

Figure 18

The LZAH algorithm used for this thesis was a modified versions of the LZSS and adaptive Huffman algorithm shown in [Nels92].

3.4.2 LZW with Adaptive Huffman Coding (LZWAH)

One method to improve the performance of the LZW algorithm for audio signals is to use a dictionary where the codewords are replaced using a least recently used [Tisc87] technique rather than throwing away the entire dictionary when it has filled up. Shamoon and Heegard [ShHe94] improve upon this method by further encoding the LZW code using adaptive arithmetic coding.

A 10 bit fixed sized codeword LZW algorithm was used in this thesis. For simplicity the dictionary was discarded when it filled up. The 10 bit codeword from the LZW algorithm was then encoded using adaptive Huffman compression rather than arithmetic coding.

Figure 19 and Figure 20 show the basic block diagrams for the LZWAH compressor and decompressor:

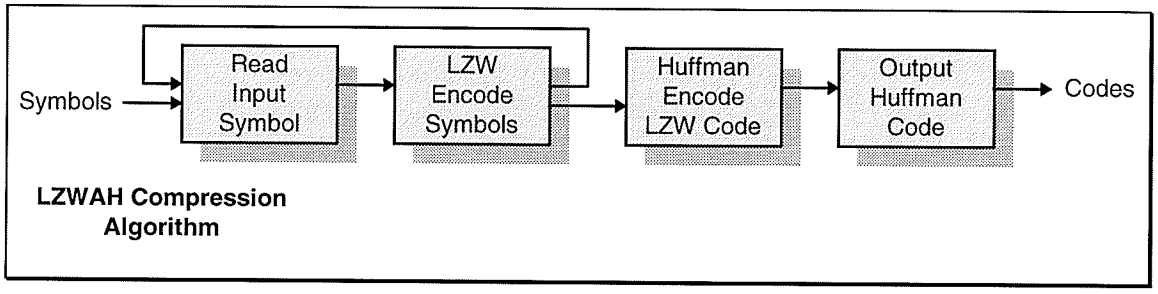

Figure 19

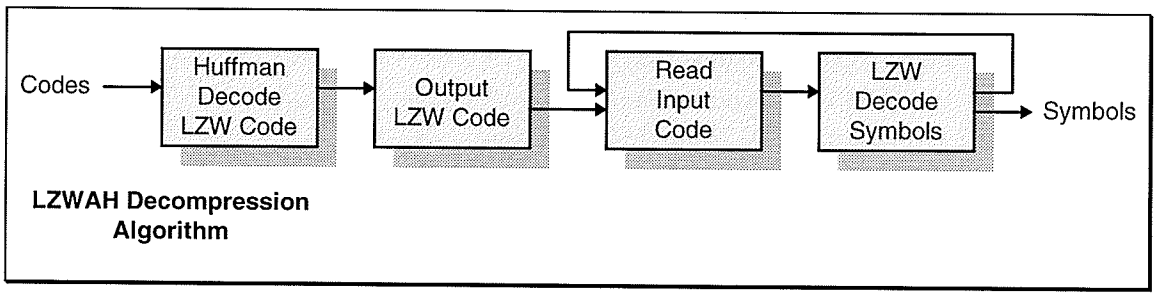

Figure 20

The LZWAH algorithm used for this thesis was a modified version of the LZW and adaptive Huffman algorithm shown in [Nels92].

Chapter 4 : Compression Algorithms and Dataset Analysis

4.1 Modem Signal Dataset

The modem and fax dataset was obtained from Jeremy Sewall at TRLabs Edmonton. The list below shows the various modem signals recorded. The modem signals were recorded using a NeXT workstation with an ISDN interface. The PCM stream was extracted directly from the ISDN DSP interface. The files were stored in MATLAB binary format as 8-bit μ-law encoded signals recorded at 8012 samples per second. The music recordings were made from FM stereo radio. The speech recordings were of the present author speaking. The audio and speech samples were recorded with 8 bit PCM at 8kHz on a Sun SPARCstation.

1. audio1 (94281 bytes) - Music 8 bits 8kHz FM stereo broadcast
2. audio2 (96129 bytes) - Music 8 bits 8kHz FM stereo broadcast
3. audio3 (90249 bytes) - Music 8 bits 8kHz FM stereo broadcast
4. audio4 (93657 bytes) - Music 8 bits 8kHz FM stereo broadcast
5. speech1 (87940 bytes) - Speech 8 bits 8kHz
6. speech2 (139103 bytes) - Speech 8 bits 8kHz
7. speech3 (73386 bytes) - Speech 8 bits 8kHz
8. v27ter_2.4_fax (1640448 bytes) - Fax 2.4k
9. v27ter_4.8_fax (1798561 bytes) - Fax 4.8k
10. v17_12.0_fax (920785 bytes) - Fax 12.0k
11. v17_14.4_fax (835425 bytes) - Fax 14.4k
12. v17_7.2_fax (1284001 bytes) - Fax 7.2k
13. v17_9.6_fax (1038177 bytes) - Fax 9.6k
14. v22bis_2.4_modem1 (201216 bytes) - Modem 2.4k
15. v22bis_2.4_modem2 (201216 bytes) - Modem 2.4k
16. v32bis_0.3_modem (465769 bytes) - Modem 300
17. v32bis_1.2_modem1 (460769 bytes) - Modem 1.2k
18. v32bis_1.2_modem2 (440769 bytes) - Modem 1.2k
19. v32bis_12.0_modem1 (201216 bytes) - Modem 12.0k
20. v32bis_12.0_modem2 (201216 bytes) - Modem 12.0k
21. v32bis_12.0_modem3 (480768 bytes) - Modem 12.0k
22. v32bis_14.4_modem (290993 bytes) - Modem 14.4k
23. v32bis_2.4_modem1 (480768 bytes) - Modem 2.4k
24. v32bis_2.4_modem2 (480768 bytes) - Modem 2.4k
25. v32bis_4.8_modem (445769 bytes) - Modem 9.6k
26. v32bis_9.6_modem1 (201216 bytes) - Modem 9.6k

27. v32bis_9.6_modem2 (201216 bytes) - Modem 9.6k
28. v32bis_9.6_modem3 (201216 bytes) - Modem 9.6k
29. v32bis_9.6_modem4 (450769 bytes) - Modem 9.6k
30. v32bis_9.6_modem5 (440769 bytes) - Modem 9.6k
31. v34_16.8_modem (721920 bytes) - Modem 16.8k
32. v34_19.2_modem (721920 bytes) - Modem 19.2k
33. v34_21.6_modem (721920 bytes) - Modem 21.6k
34. v34_24.0_26.4_modem (721920 bytes) - Modem 24.0k/26.4k
35. v34_24.0_modem (721920 bytes) - Modem 24.0k
36. v34_26.4_modem (721920 bytes) - Modem 26.4k
37. v34_28.8_modem (721920 bytes) - Modem 28.8k

Figure 21 shows the μ-law curve used in PCM. Table 11 shows the amplitude levels of the normalized input and output analog signals that correspond to the various unsigned codes shown in the μ-law curve.

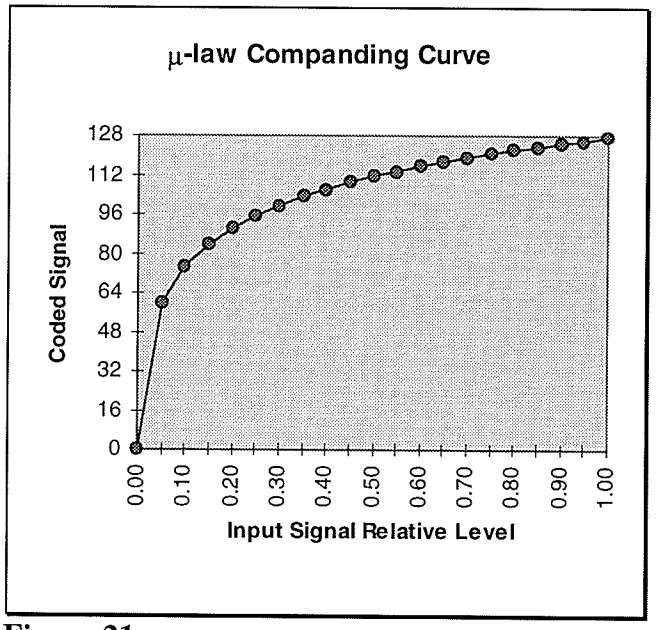

Figure 21

Unsigned Code	Input	Output
127	1.0	1.0
112	0.5	0.875
96	0.25	0.752
80	0.125	0.630
64	0.0625	0.510
48	0.0313	0.396
32	0.0156	0.290
16	0.0078	0.198
0	0	0

Table 11

4.2 Statistical Compression Results

The statistical coders do a much better job at compressing the data than the dictionary coders do. They are able to take advantage of the short term first-order signal statistics. The dictionary coders try to take advantage of string repetitions over the short term. There appears to be little if any repetition over the short term which accounts for the poor performance of the dictionary coders.

Figure 22 shows the complete 8 bit symbol frequency count for the digitized v.34 28.8k modem signal. The 28.8k modem sample was 721920 bytes long which is equivalent to

approximately 90 seconds. The graphs shows a surprising amount of regularity. The different frequency count levels can be traced back to the quantization levels of the μ-law curve.

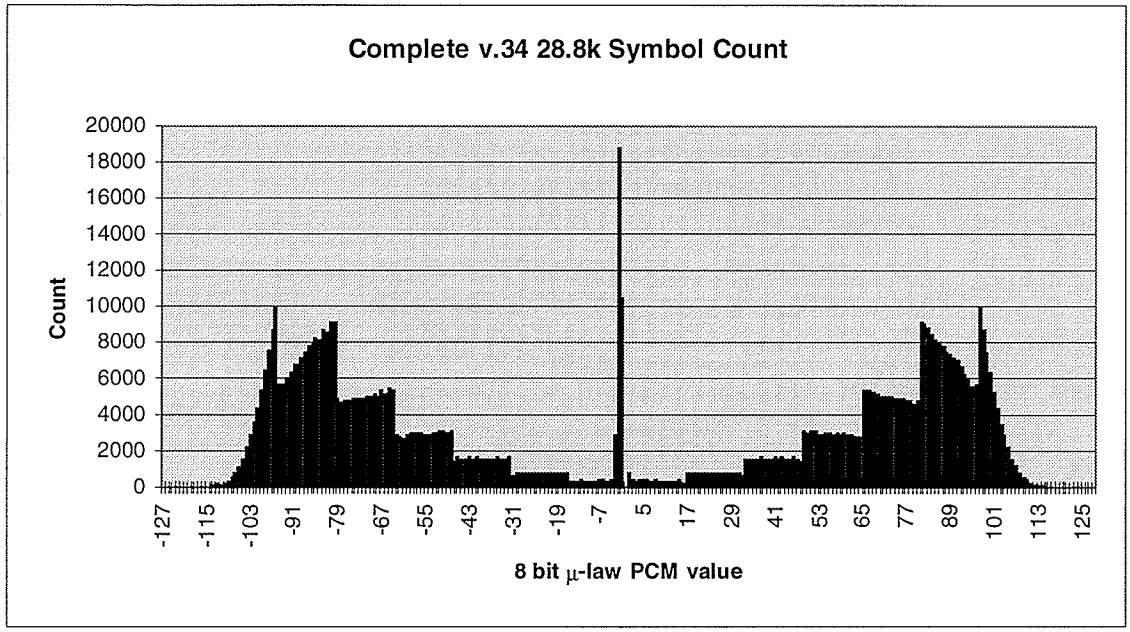

Figure 22

Figure 23 and Figure 24 show the symbol frequency counts for two 5000 digitized sample plots adjacent in time.

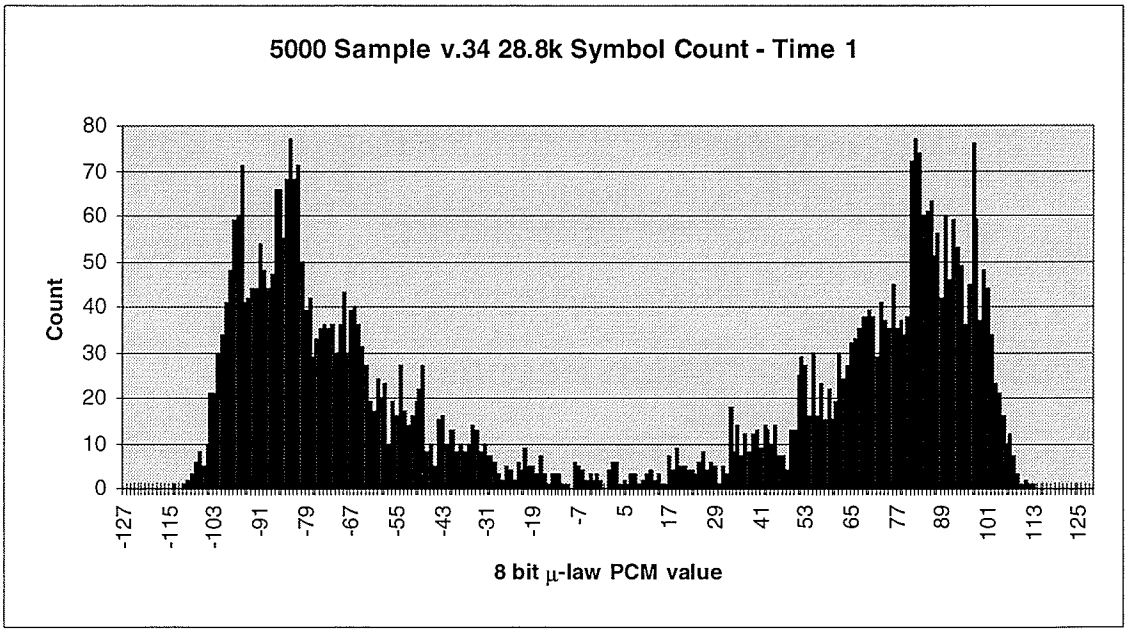

Figure 23

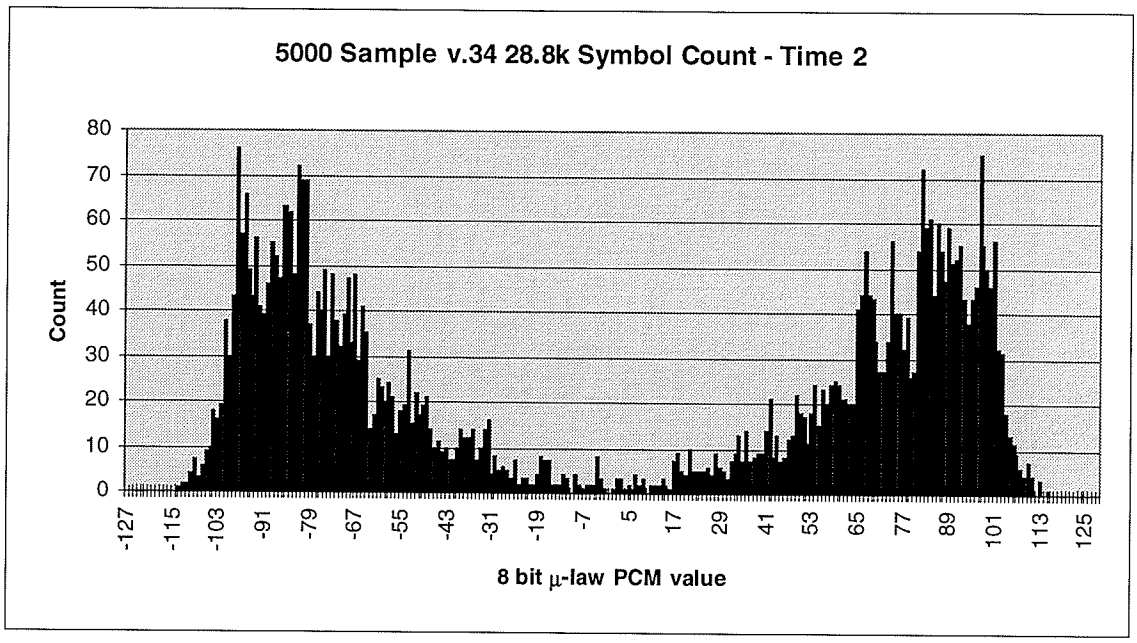

Figure 24

Although the complete plot shows a high degree of regularity, the degree of regularity in the two 5000 sample sets are not as high. Notice that the statistics of the symbols change signficantly between the two figures. The changes in statistics would imply that for a statistical compression algorithm to achieve significant compression, it would have to adapt fairly rapidly to the changes in the input symbol statistics.

4.2.1 Adaptive Huffman

Figure 25 shows the results for adaptive Huffman compression with rescaling of the Huffman tree after the maximum weight reaches 32768. Adaptive Huffman compression was capable of encoding the 8 bit μ-law input symbols with approximately 6.5 to 7.6 bits per input symbol.

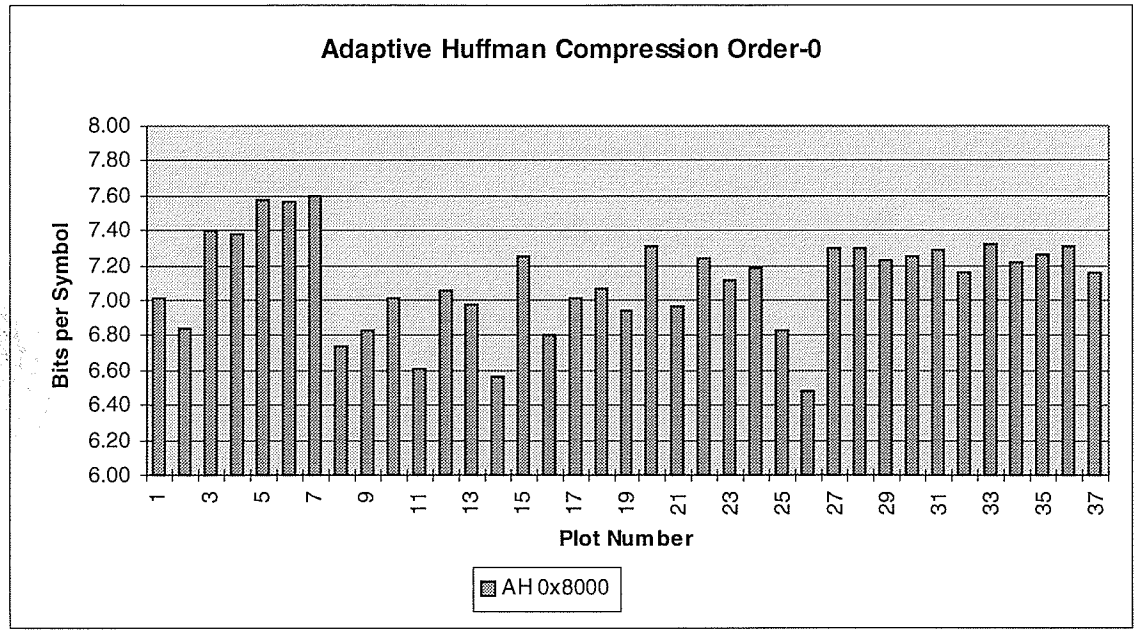

Figure 25

Figure 26 shows the results obtained by the adaptive Huffman algorithm with rescaling of the Huffman tree at:

- 0x1000 (4096) maximum weight
- 0x2000 (8192) maximum weight
- 0x4000 (16384) maximum weight

It shows that the adaptive Huffman algorithm achieves slightly better compression with most samples in the dataset when the tree is rescaled more frequently. As mentioned previously, this would indicate that the order-0 statistics of the various signals in the dataset change fairly rapidly, usually within 5000 samples. The disadvantage of rescaling the tree more frequently is the computational costs involved.

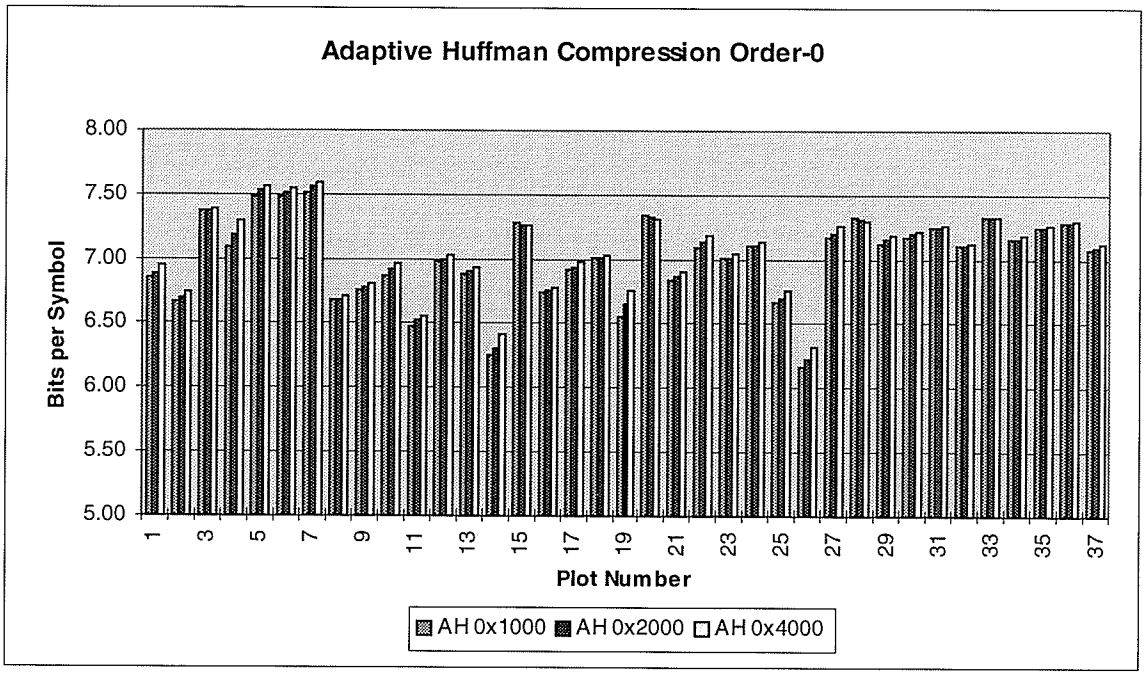

Figure 26

4.2.2 Adaptive Arithmetic

Figure 27 shows the results for adaptive arithmetic compression. It was capable of encoding the 8 bit μ-law input symbols with approximately 6.25 to 7.5 bits per input symbol. The adaptive arithmetic algorithm was able to achieve slightly better compression than the adaptive Huffman algorithm but this was generally only a savings of a few bytes.

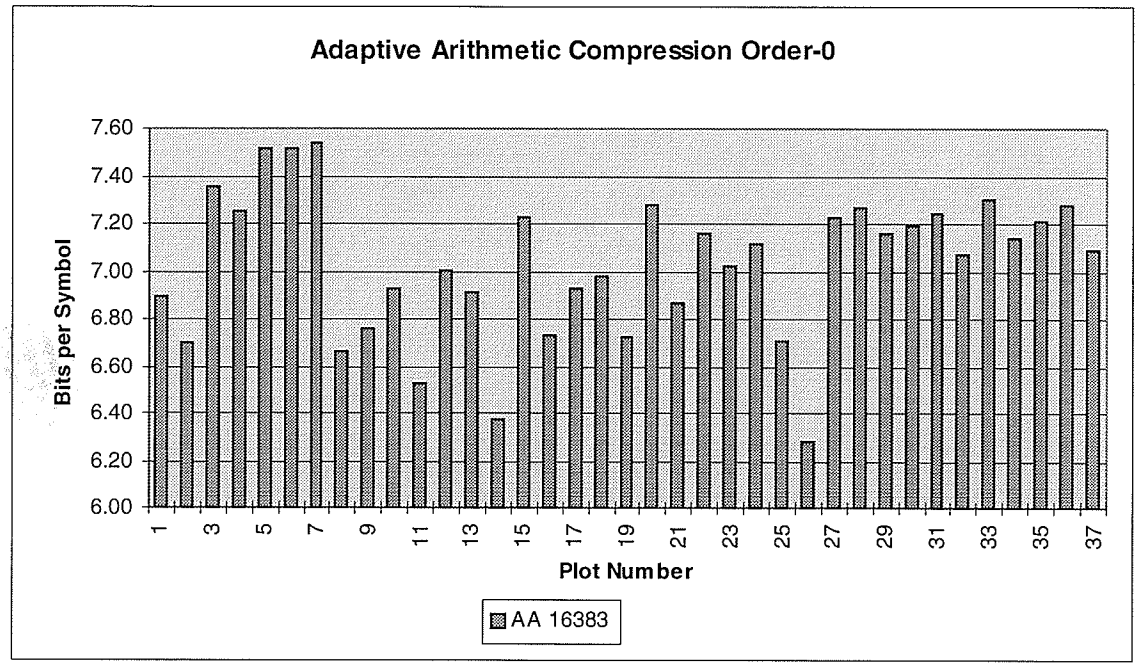

Figure 27

Figure 28 shows the compression ratios for the adaptive arithmetic coder. The graph lists the results for the adaptive arithmetic algorithm with rescaling of the symbol table at:

- 0x1000 (4096) maximum weight
- 0x2000 (8192) maximum weight
- 0x4000 (16384) maximum weight

The adaptive arithmetic algorithm exhibits properties similar to the adaptive Huffman algorithm in that it achieves better compression with most samples in the dataset when the symbol table is rescaled more frequently. As with the Huffman algorithm, rescaling the symbol table counts more frequently adds to the computational costs.

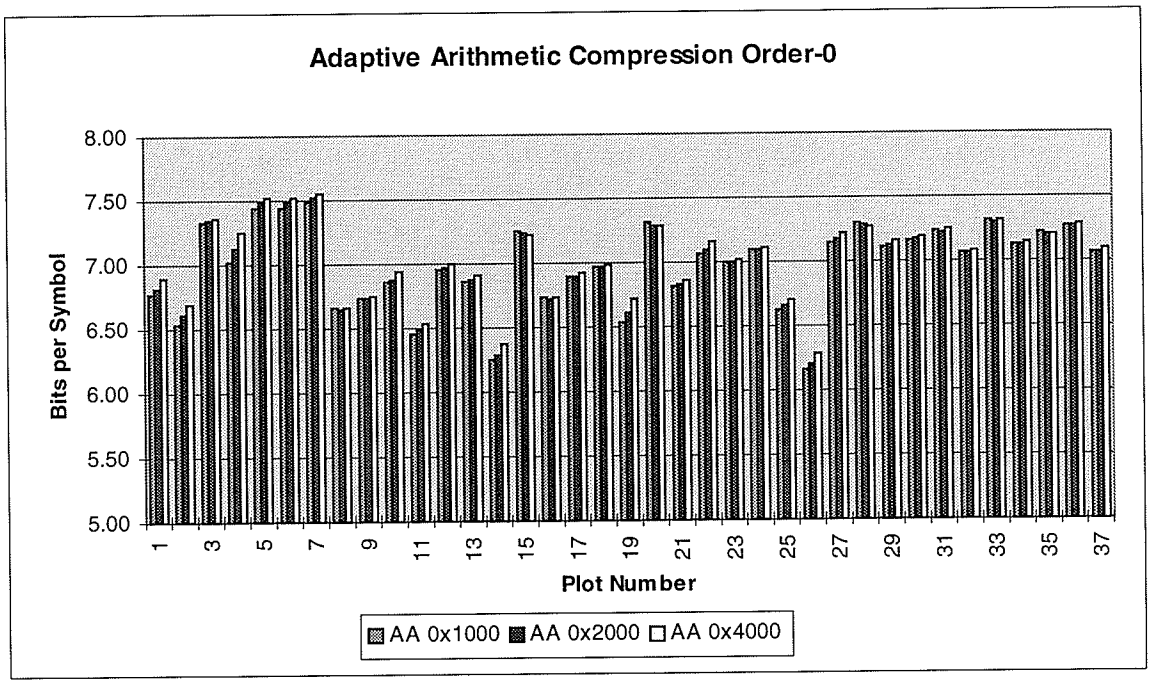

Figure 28

Figure 29 shows a comparison of Huffman compression versus arithmetic compression with codebook rescaling every 4096 (0x1000) input symbols. Arithmetic coding is typically capable of encoding the data samples with 0.1 to 0.01 fewer bits per input symbol. This is not a sufficient gain compared to the increased amount of computation required for arithmetic coding.

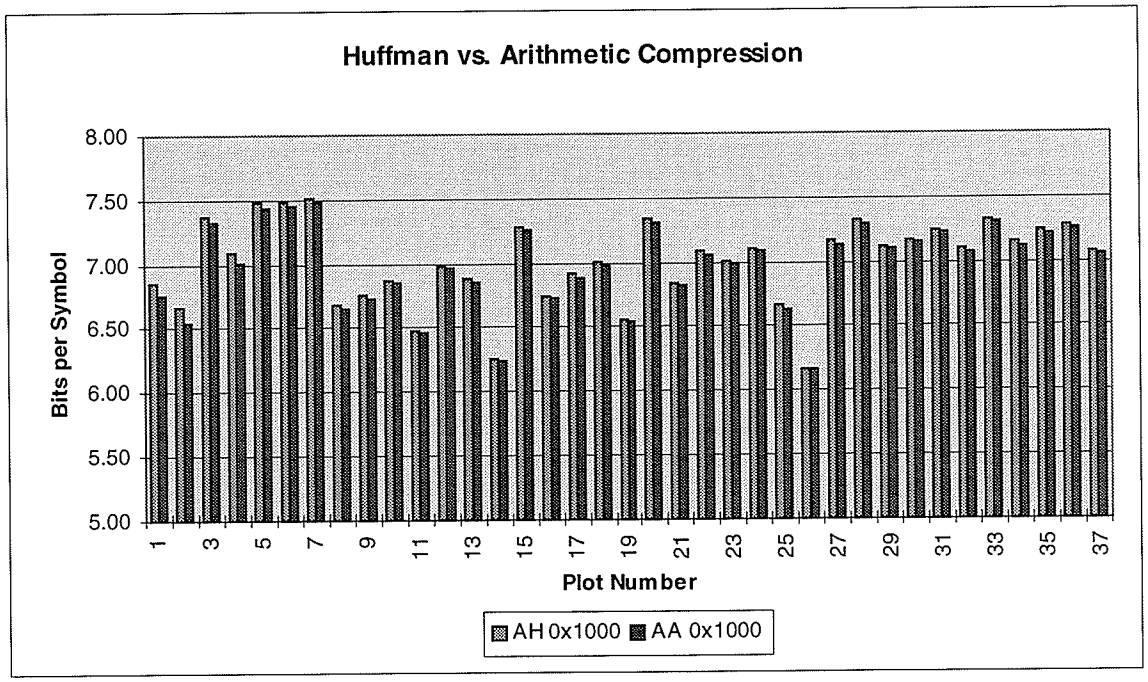

Figure 29

4.3 Dictionary Compression Results

4.3.1 LZW

Figure 30 shows the results for the fixed codebook LZW algorithm. In the majority of the data samples, the LZW algorithm required up to 10.75 bits to encode the 8 bit μ-law input symbols. The LZW algorithm compressed data samples 2, 14, 16, and 26 but expanded the rest. Using a smaller dictionary improves the compression ratio in most cases.

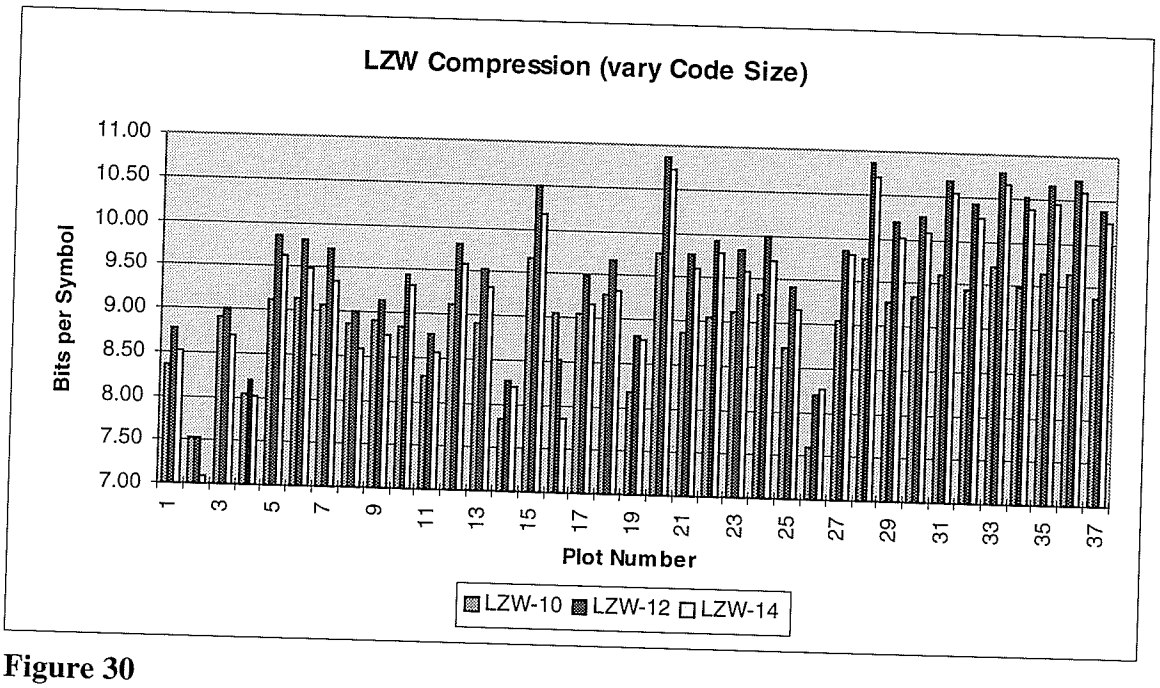

Figure 30

Figure 31 shows the results for the variable codeword LZWV algorithm. In some cases the LZWV algorithm was able to compress the data samples (1, 2, 3, 4, 8, 9, 11, 14, 16, and 26). This time the larger codebook (LZWV-15) had a better performance than the smaller codebooks (LZWV-13 and LZWV-11). The LZWV algorithm also performed better than the LZW algorithm but still managed to expand more of the data samples then it compressed.

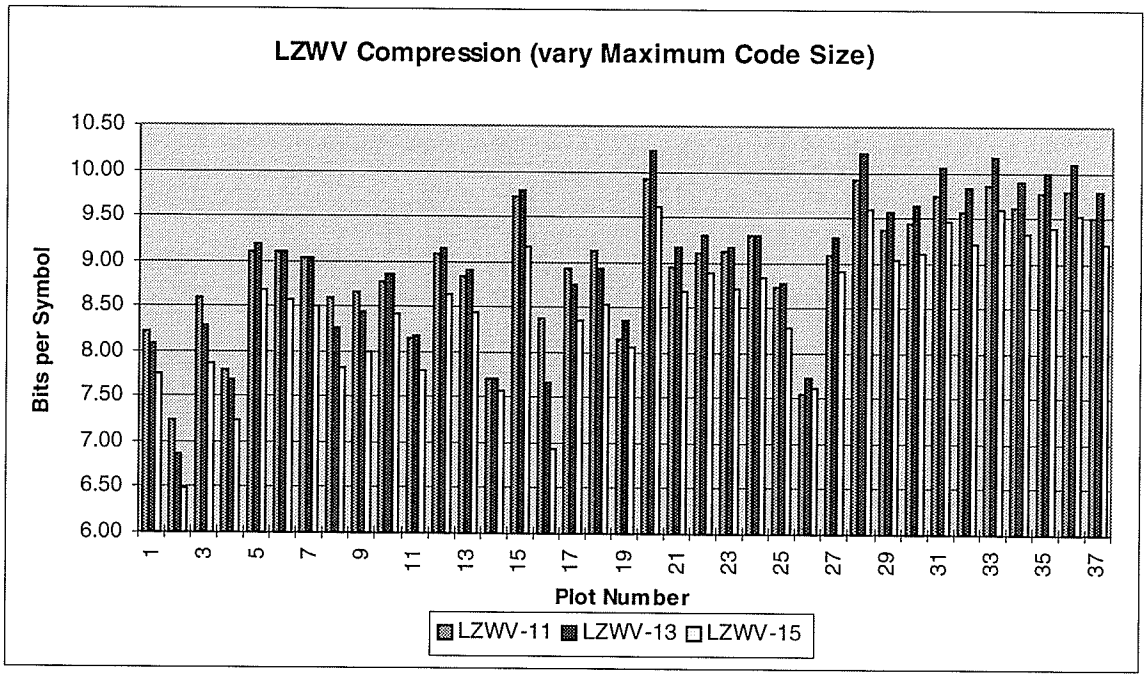

Figure 31

The LZW and LZWV algorithms failed to compress most of the data samples. The main problem is that the dictionaries are simply thrown away when they are filled. Any previous modeling data contained in those dictionaries are lost and a new model has to be built. One method to solve this problem would be to discard the least recently used and least frequently used phrases when the dictionary fills [Tisc87]. This method is expected to work better than simply throwing the dictionary away when it is full. The disadvantage of this method is the increase in computation requirements for maintaining the dictionary. The statistical compression algorithms showed that not all 256 (8 bit) input symbols were used. Another possible improvement to the LZW algorithm would be not to preload the dictionary with all 256 input symbols.

The next three graphs show various statistics of the 12 bit fixed codeword LZW algorithm. With 12 bit codewords the dictionary is capable of holding 4096 codes, composed as follows:

- 256 codes for the 8 bit input symbols
- 1 code for the end-of-stream symbol
- 3839 usable match codes
- 4096 codes total

Figure 32 shows the average number of input symbols read before the dictionary was cleared. The plot shows that the algorithm fills up the dictionary very quickly since it is cleared after at most 6000 input symbols, and on average after 5000 input symbols.

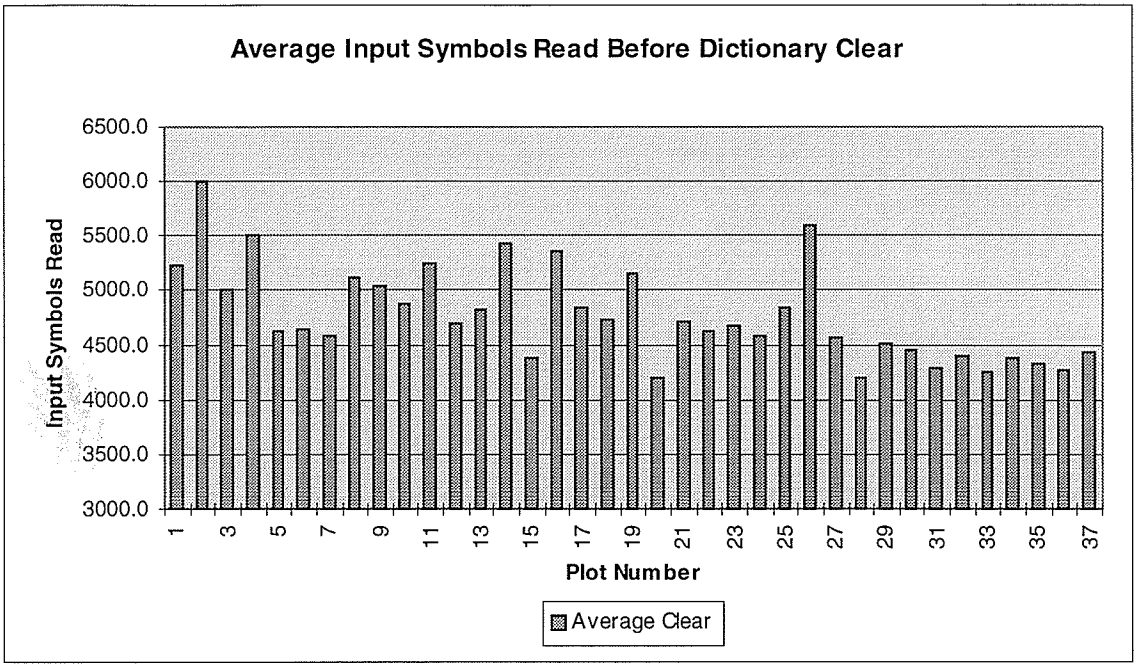

Figure 32

Figure 33 shows the average normalized entropy of each dictionary used by LZW for the entire dataset. It shows that some codewords entered into the dictionary are not utilized. This is one reason why LZW requires more than 8 bits on average to encode the input symbols.

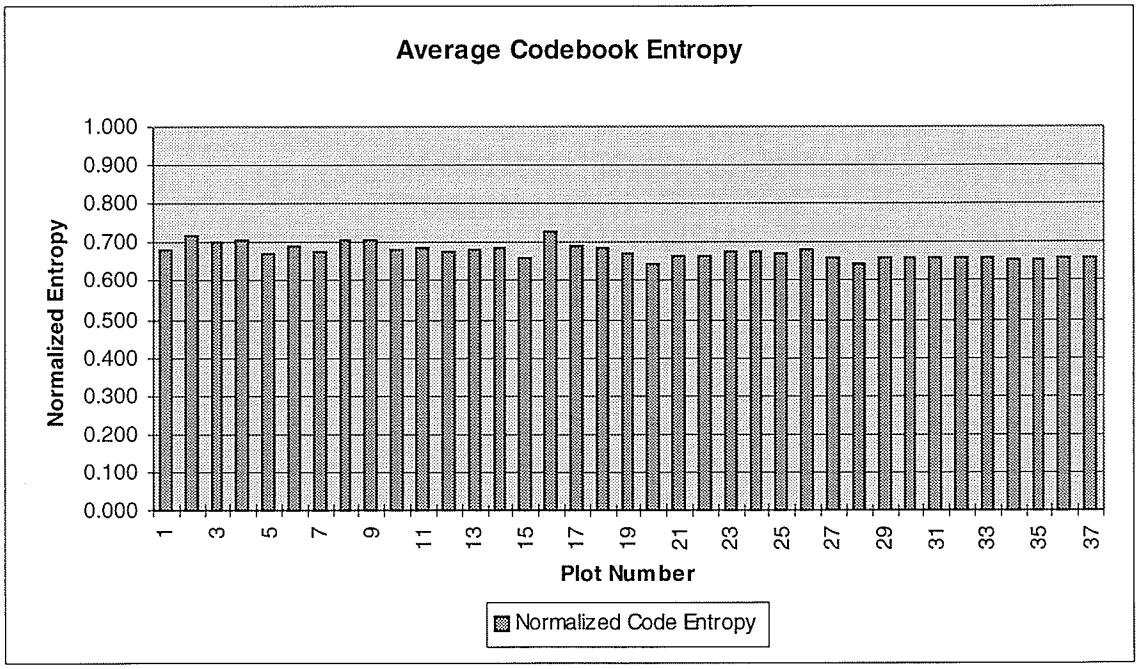

Figure 33

Figure 34 shows the average number of codes emitted per input symbol. Since the LZW algorithm uses 12 bit codewords to represent 8 bit input symbols, it must represent at least 1.5 input symbols per codeword to achieve data compression. The graph shows that this only occurs with data sample 2 (audio2 signal).

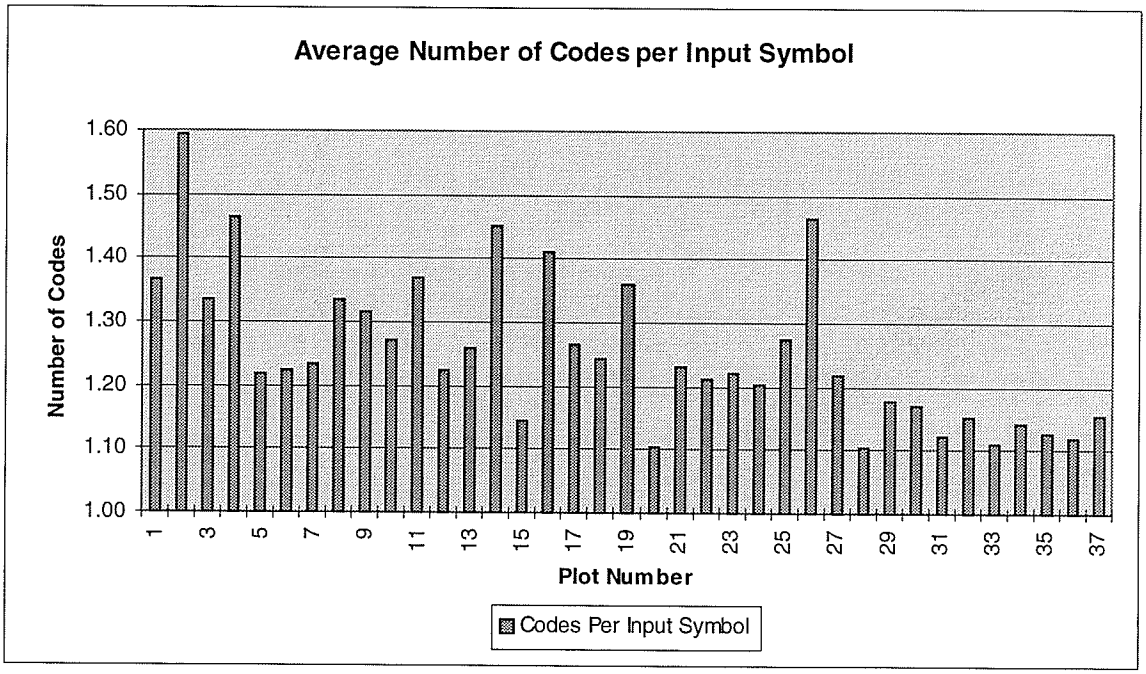

Figure 34

4.3.2 LZSS

Figure 35 shows the results for the LZSS algorithm using 4 bits for the match length and varying the index bits from 10 bits to 14 bits. In general the LZSS algorithm fared better in encoding the 8 bit μ-law input symbols than LZW but it nevertheless required 8 to 9 bits to encode the input symbols in some cases. Having a larger windows size does not appear to have a significant impact on the bits required to encode the input symbols.

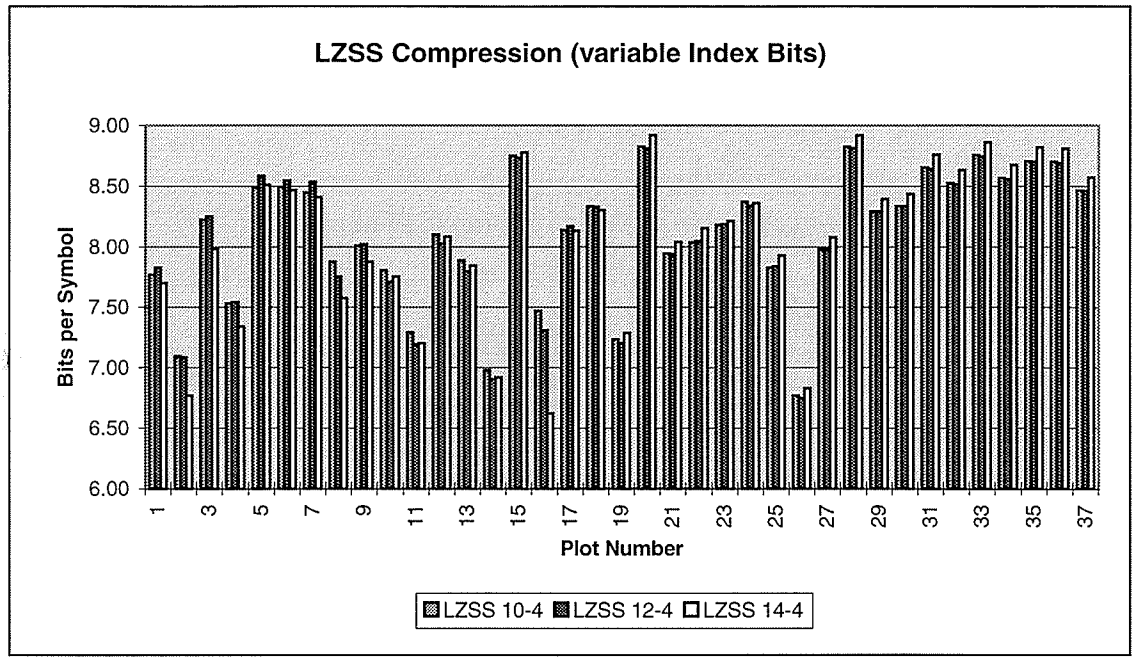

Figure 35

Figure 36 shows the results for the LZSS algorithm using 12 bits for the index and varying the match length from 2 bits to 4 bits. Higher compression was achieved using 2 bits for the match length. This would indicate that when a match is found that match length is usually a small number rather than a larger one.

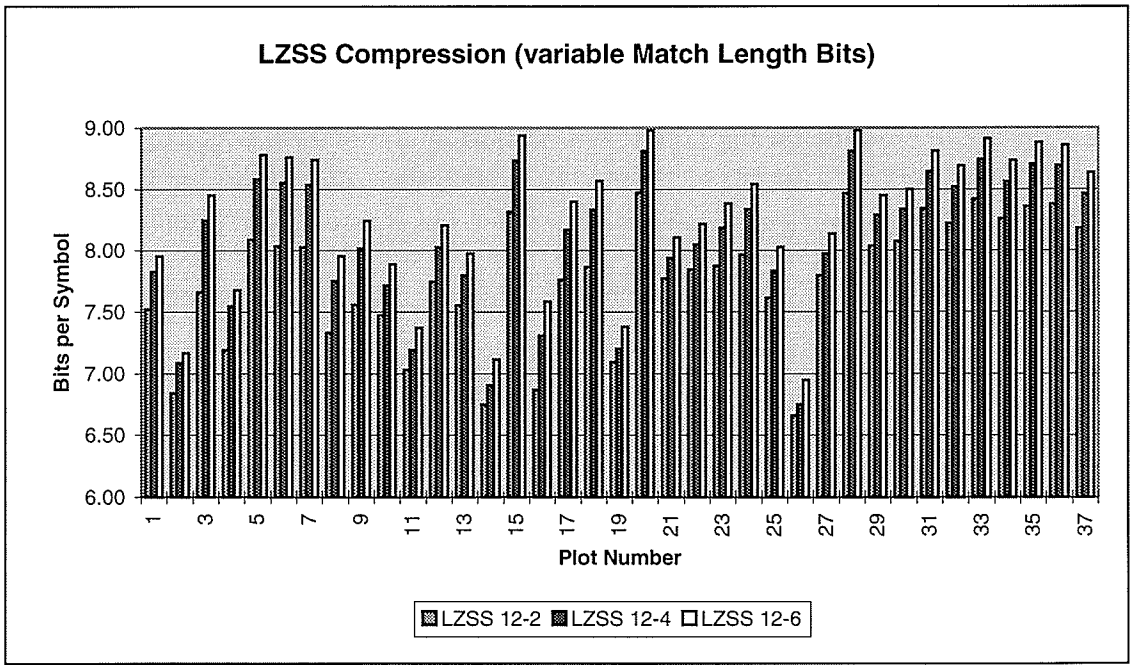

Figure 36

The next set of graphs will examine the compression performance of the LZSS algorithm using a 12 bit window and 4 bit match length. Using these parameters, the break-even value for match pointers is 2 input symbols. Therefore a 4 bit match length can be used to encode matches of 2 to 17 symbols. This means that two input symbols can be encoded in fewer bits than two literals. The table lists the number of bits required to encode various numbers of input symbols.

Number of Input Symbols	1	2	3	n
Number of bits to encode Input Symbols	8	16	24	8n
Number of bit to encode as LZSS Symbol	9	18	27	9n
Number of bits to encode as LZSS Match Pointer	17	17	17	17

Table 12

Table 12 shows that for LZSS to achieve compression, it must encode as many input symbols as possible using match pointers of length 3 or more. With a break-even value of 2 input symbols, a 4 bit match length can be used to encode match lengths ranging from 2 to 17 input symbols in the text window.

Figure 37 shows a comparison of the percentage of input symbols encoded as LZSS match pointers versus symbols. In general, those samples in the dataset, that the LZSS algorithm was capable of encoding using more match pointers, achieved better compression.

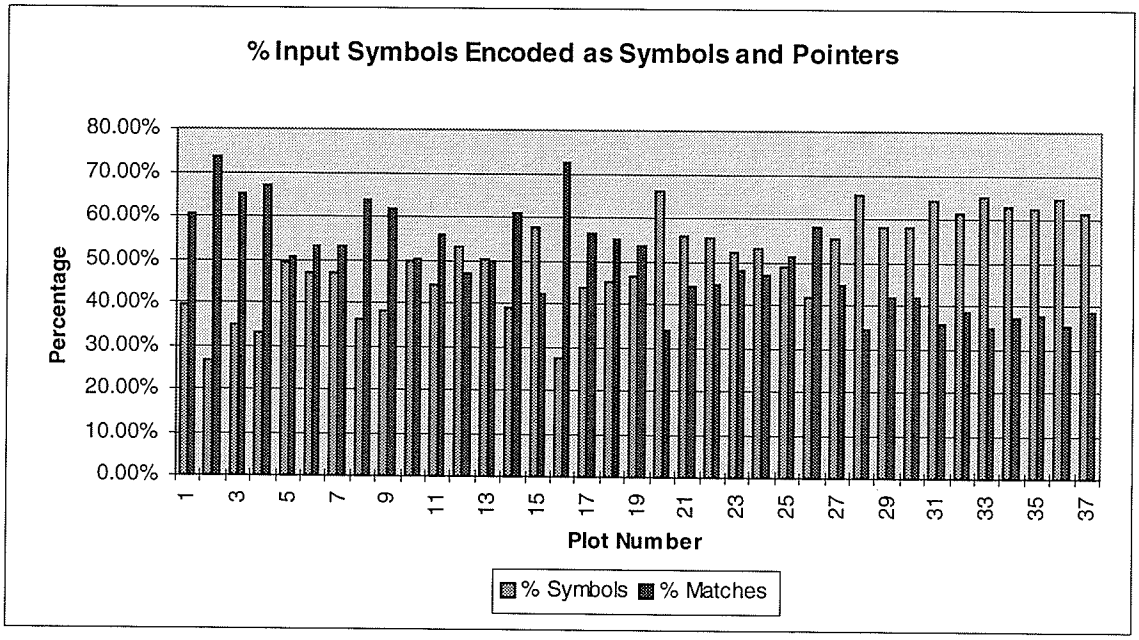

Figure 37

Figure 38 shows the percentage of match pointers with a match length of 2 versus match lengths greater than 2. A match length of 2 input symbols is the most common.

However it still requires more bits to encode a match of length 2 (17 bits) than is required to encode 2 input symbols (16 bits). One method to improve this would be to use static Huffman encoding for the match length bits. This is done with the LZH algorithm discussed in the next section.

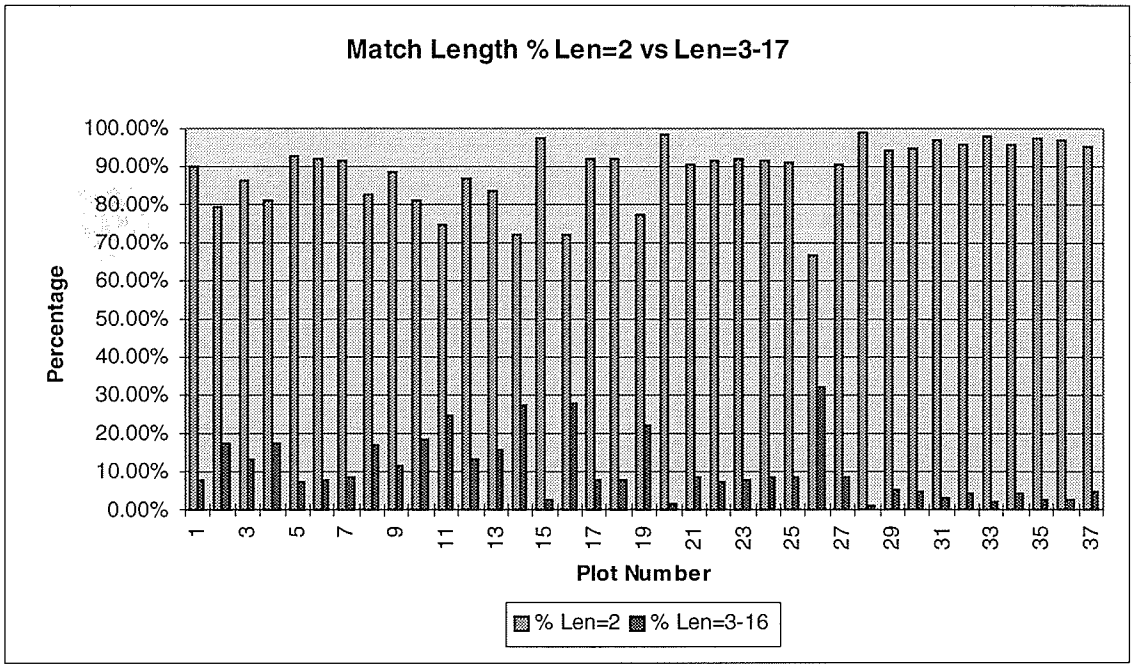

Figure 38

Figure 39 shows the normalized entropy for the pointer position. The lowest entropy is still over 0.9. This indicates that the pointer position is evenly distributed throughout the sliding window.

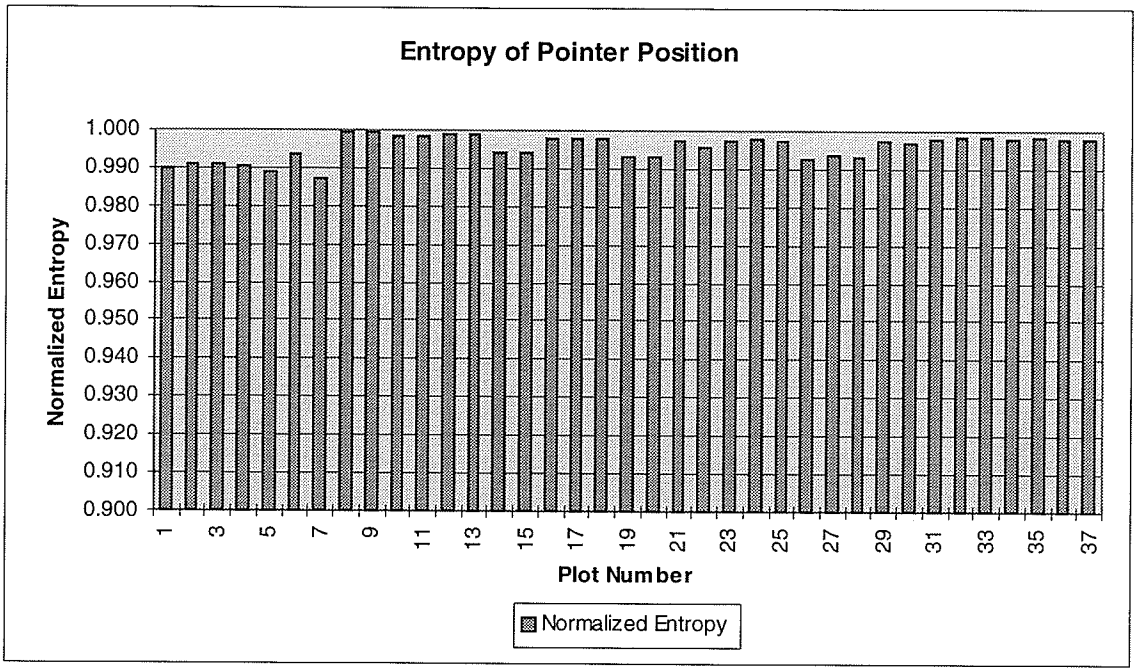

Figure 39

4.3.3 LZH

This section shows the results for LZSS using static Huffman encoding for the match length bits. This algorithm is called LZH and is discussed in the paper [Bren87]. The Huffman coding of the match length allows short matches to be encoded using fewer bits.

Number of Input Symbols	1	2	3	...	5	...	n
Number of bits to encode Input Symbols	8	16	24	...	40	...	8n
Number of bit to encode as LZH Symbol	9	18	27	...	45	...	9n
Number of bits to encode as LZH Match Pointer	-	14	16	...	18	...	20

Table 13

Figure 40 shows the results of the LZH algorithm using a 12 bit sliding window and Huffman codes to represent a 4 bit match length. LZH was capable of encoding most of the data samples in fewer than 8 bits. This is significantly better than the LZSS algorithm.

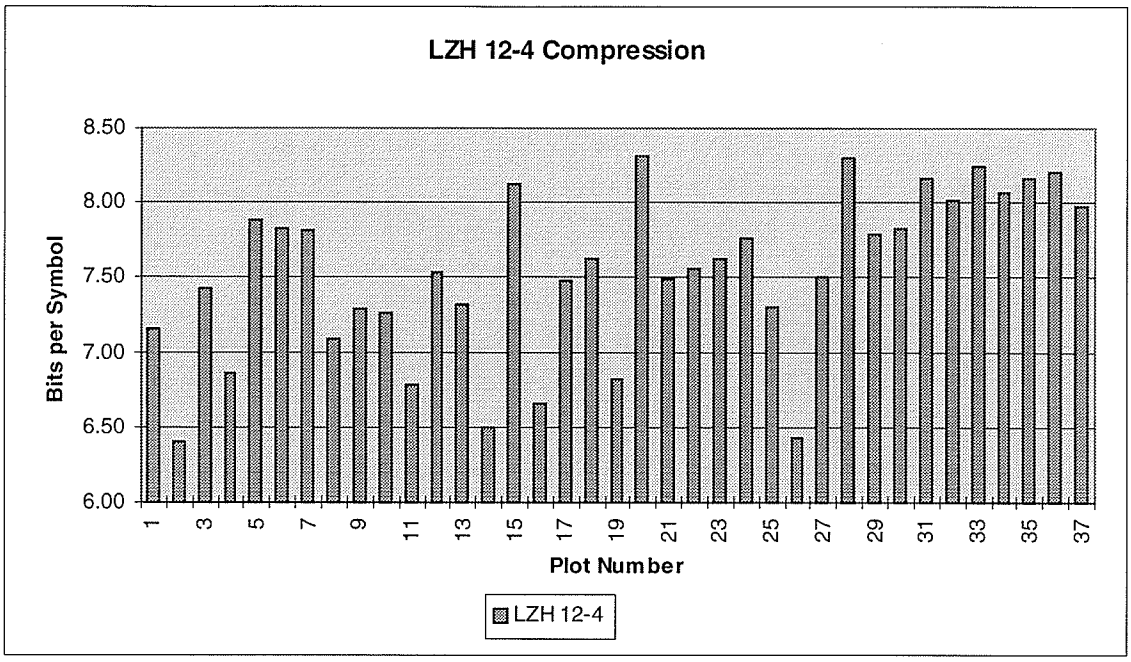

Figure 40

4.3.4 LZFGH

Figure 41 shows the results for LZFGH compression using 12 bits for the match position pointer and a 4 bit Huffman coded match length. LZFGH was capable of encoding most of the data samples with fewer than 8 bits per input symbol.

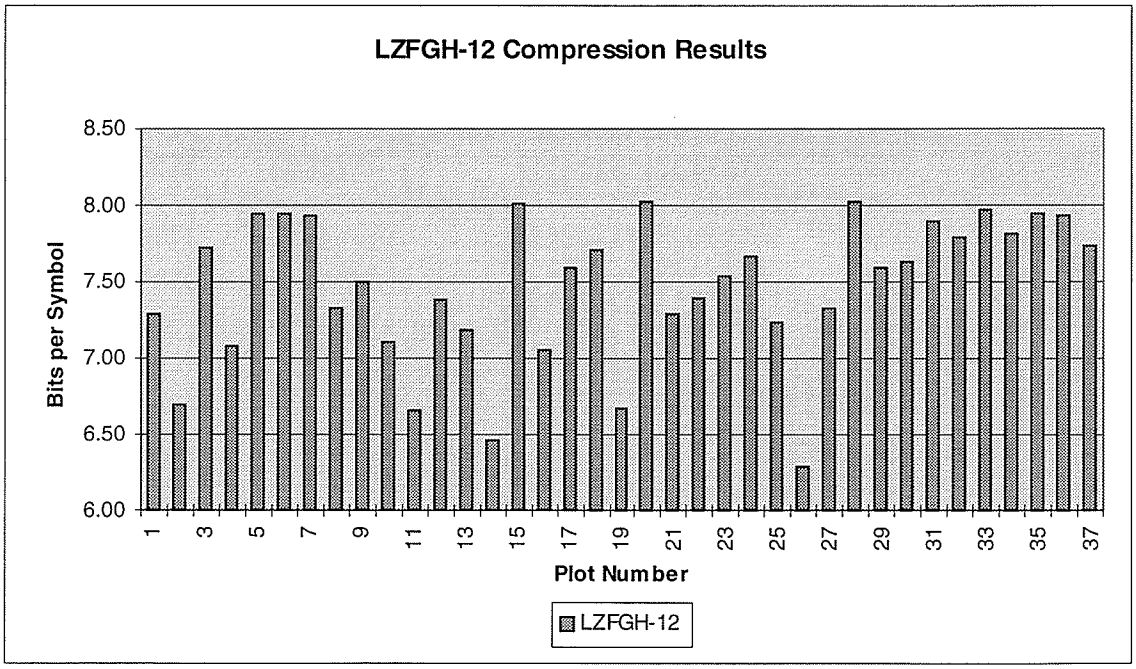

Figure 41

Figure 42 shows a comparison of the percentage of input symbols encoded as literals versus match pointers. The LZFGH algorithm encodes fewer input symbols as match pointers compared to LZSS. This is mainly due to the fact that LZFGH is capable of encoding strings of symbols very close to 8 bits per symbol (compared to 9 bits used per literal by LZSS) if it encodes longer strings of literals. This result is that fewer input symbols are encoded as match pointers.

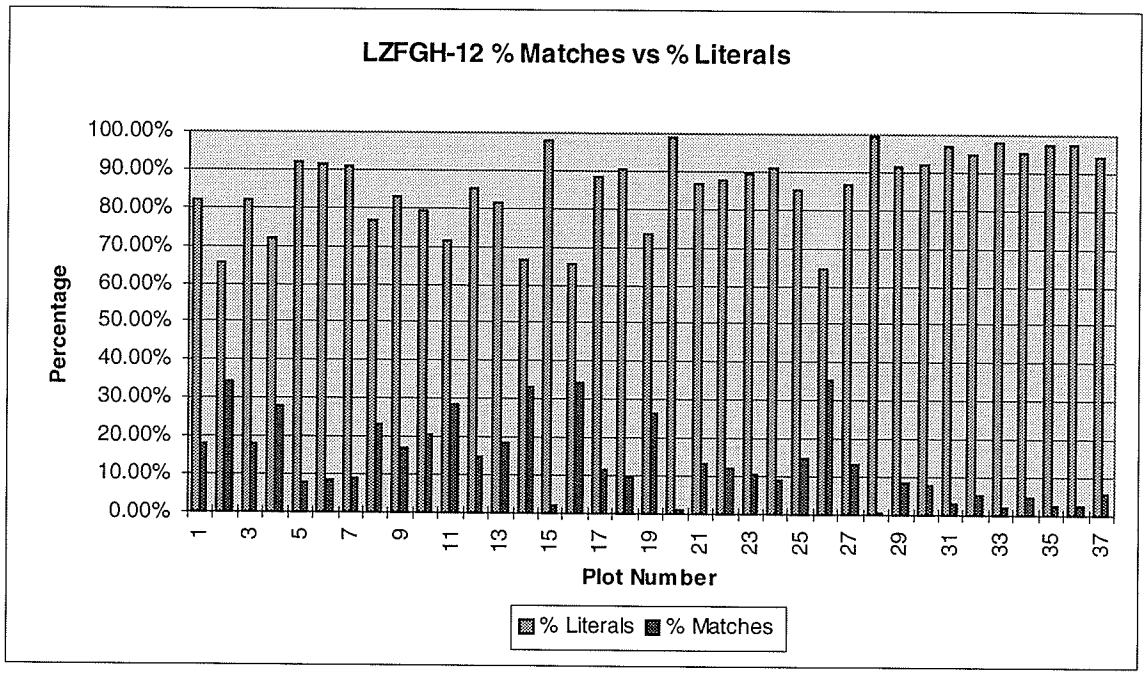

Figure 42

Table 14 shows the number of bits required to encode input symbols, LZFG literals and LZFG match pointers using a window size of 12 bits and a match length of 4 bits. The break-even value was determined by trial and error. It was discovered that a break-even value of 3 input symbols allowed LZFG to encode longer strings of literals close to the 8 bits per input symbol. With a break-even value of 3 input symbols, any input symbols encoded using match pointers requires fewer bits than encoding them as 8 bit values.

Number of Input Symbols	1	2	3	...	10	...	n
Number of bits to encode Input Symbols	8	16	24	...	80	...	8n
Number of bit to encode as LZFG Literal	9	18	27	...	90	...	10+8n
Number of bits to encode as LZFG Match Pointer	-	-	15	...	20	...	17

Table 14

Figure 43 shows the average number of bits required to encode all LZFG literals for the entire dataset. The worst case is over 8.3 bits to encode literals for data sample 16. This is much less than the 9 bits required by LZSS.

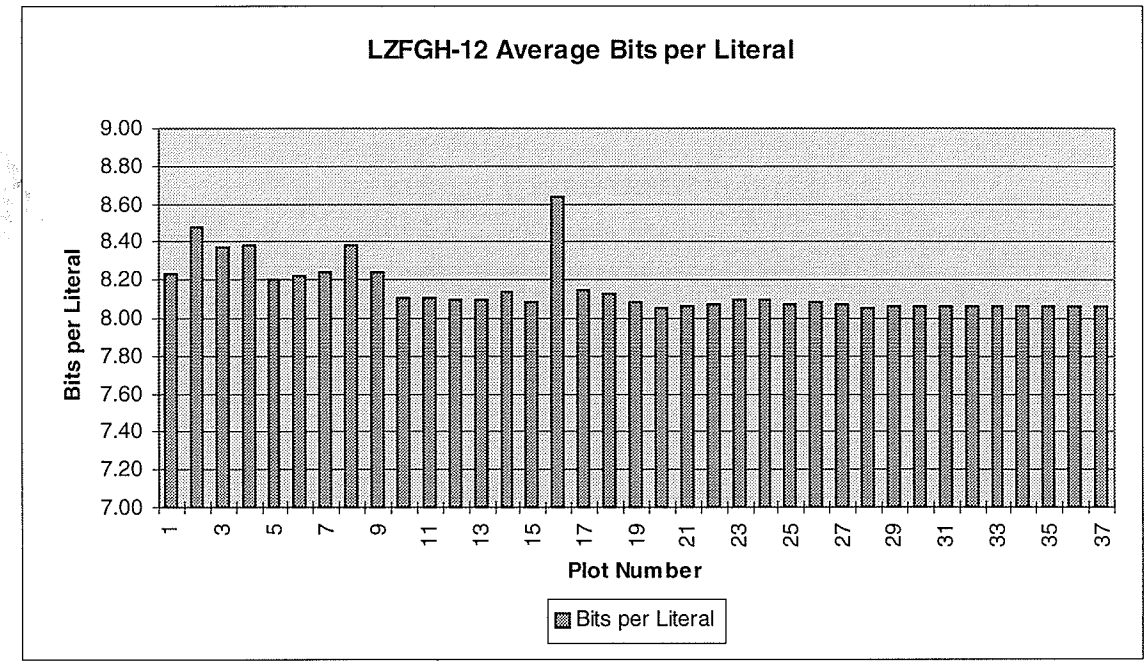

Figure 43

4.4 Dictionary-Statistical Compression Results

4.4.1 LZAH

Figure 44 shows the compression ratios achieved for LZAH using a window size of 12 bits and a match length of 4 bits. The LZAH algorithm was able to encode the 8 bit input symbols from about 6 bits to 7.5 bits per symbol. This is a slight improvement over adaptive Huffman algorithm.

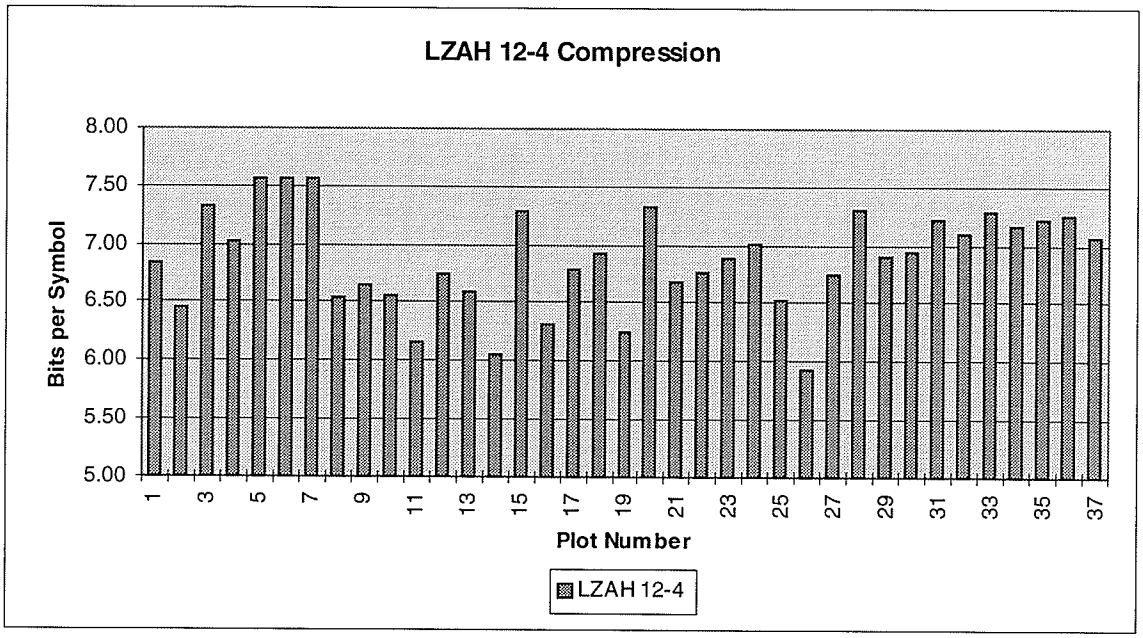

Figure 44

Figure 45 shows the percentage of input symbols that are encoded as match pointers versus Huffman encoding. The LZAH algorithm spends most of its time encoding symbols using adaptive Huffman rather than using the match pointer. The LZAH algorithm is capable of encoding the actual input symbols in fewer bits using adaptive Huffman than using match pointers. The break-even value is 3 input symbols. Therefore, the LZAH algorithm will only encode the input symbols using match pointers if it can find matches of length 3 or longer. As was shown with the LZSS statistics over 70% of the matches found were of length 2. Since there are fewer matches of length 3 or more, the LZAH algorithm would use fewer match pointers. In general, the more input symbols that can be coded, using match pointers with long match lengths, the better the compression ratio.

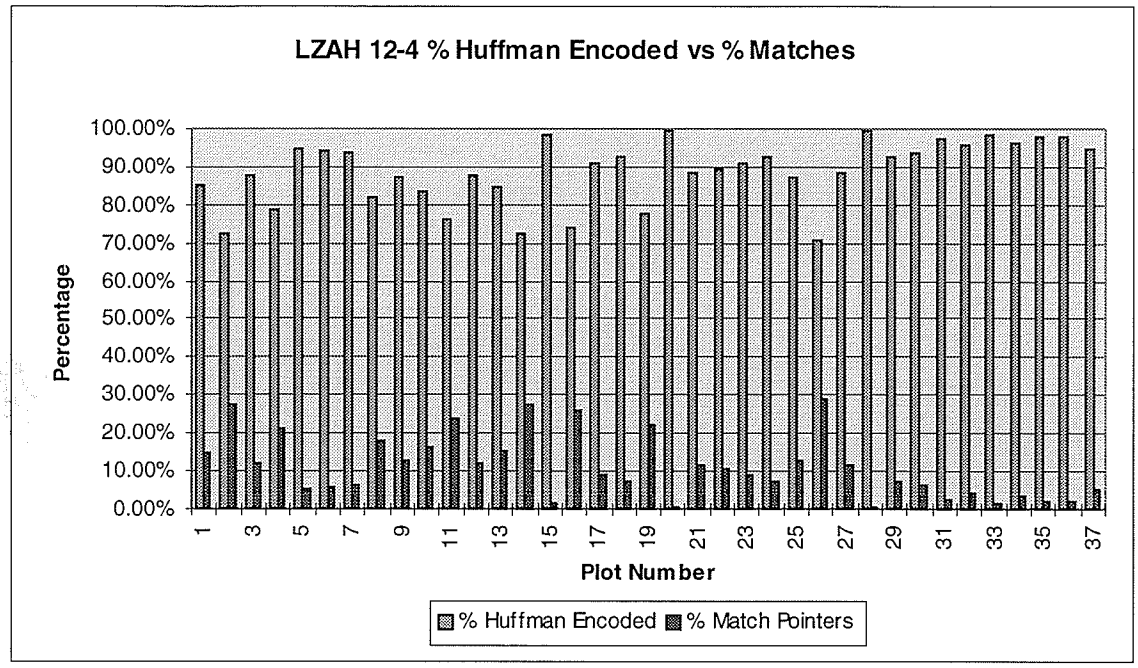

Figure 45

4.4.2 LZWAH

Figure 46 shows the results of the LZWAH algorithm compared to adaptive arithmetic compression. The LZW component used 10 bit codewords. The adaptive Huffman component encoded these as 10 bit codewords. The LZWAH improves the compression ratio slightly. Two factors hinder the performance of the LZWAH algorithm. The first is that the dictionary is simply discarded when it is filled. A least-recently used or least-frequently used algorithm for pruning the dictionary could be used to improve the encoding efficiency of the LZW agorithm. However, this would also come at a significant computational cost. The second factor is the large number of symbols that must be encoded by the adaptive Huffman compressor. While this does not impact the compression efficiency, it also has a very high computational load for maintaining the Huffman tree.

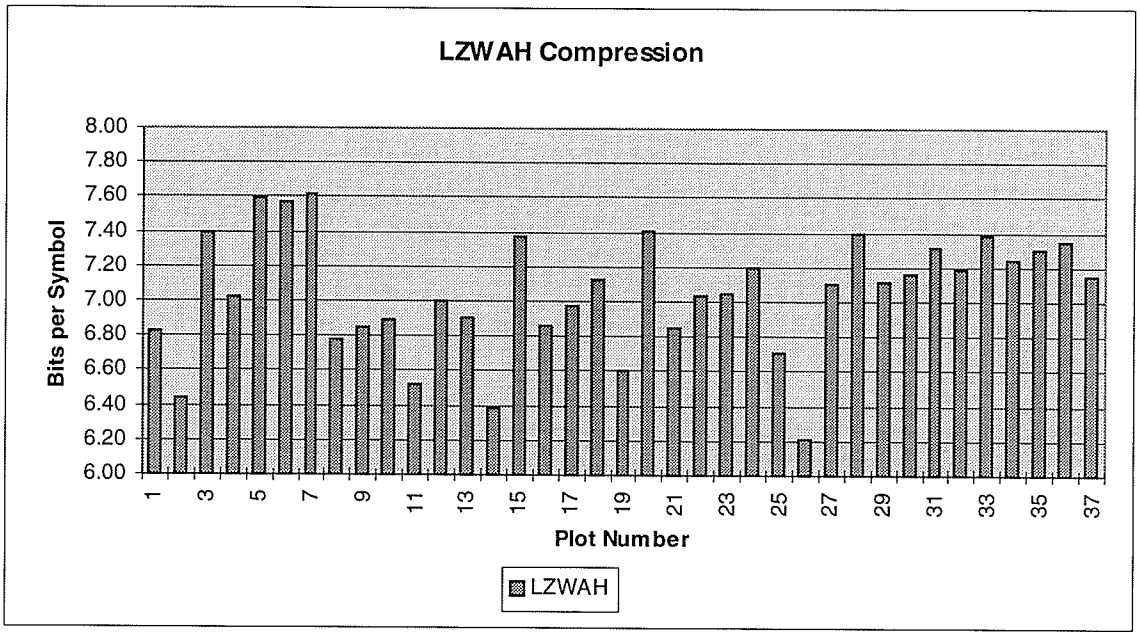

Figure 46

4.5 Summary of Data Compression Results

Figure 47 shows the average number of bits needed to encode the entire dataset of digitally-sampled modem and audio signals for all the compression algorithms investigated. The best results show that the 8 bit μ-law input symbols can be encoded using about 7 bits with any of the statistical or dictionary-statistical compression algorithms.

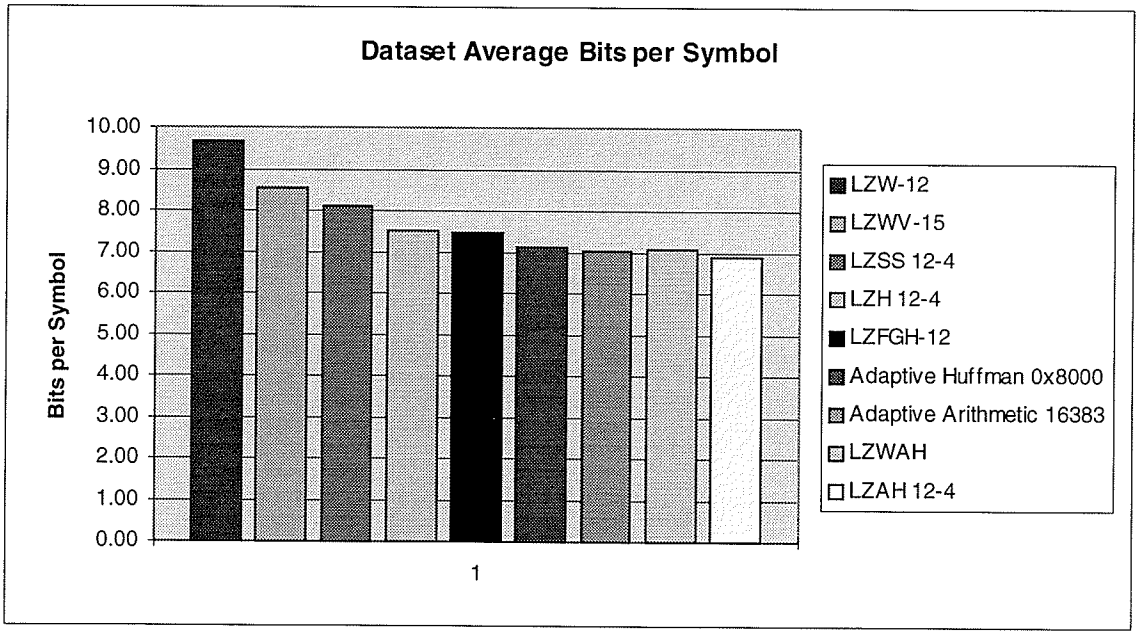

Figure 47

Figure 48 shows the compression results sorted by modem speed. It seems logical to conclude that lower speed modem signals would have more redundancy and would achieve better compression results. However this is not the case. Any redundancy is lost in the conversion of a digital bit stream into an analog signal by a modem and then digitally sampling this analog signal at the telephone network.

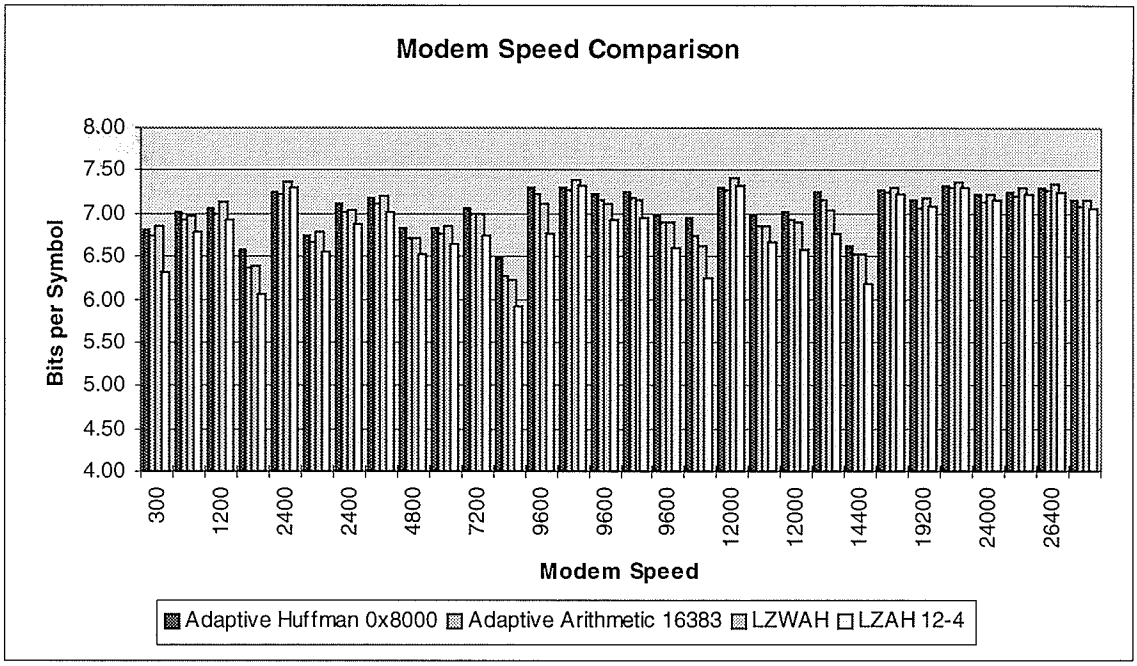

Figure 48

Chapter 5 : Performance Analysis

5.1 Algorithm Comparison

Chapter 3 showed that the following four compression algorithms were able to consistently compress the modem dataset:

- Adaptive arithmetic compression
- Adaptive Huffman compression
- LZAH
- LZWAH

These four compression algorithms were applied to the modem dataset three times and the computational performance statistics were gathered by the GNU profiler. The chapter will analyze the performance of these four algorithms using the profiler statistics.

The four compression algorithms were run using the following:

- 85MHz Sun SPARCStation 5
- 64MB RAM
- GNU C Compiler GCC v2.7.0 (-O3 Optimization)
- GNU Profiler gprof version 5.6 (Cygnus)

The following parameters were used for each compression algorithm:

Adaptive Arithmetic:

- Rescale Arithmetic Symbol Table at count 16383

Adaptive Huffman:

- Rescale Huffman Tree at maximum weight of 32768

LZAH:

- Rescale Huffman Tree at maximum weight of 32768
- 12 bit Window Size (4096 bytes)
- 4 bit Look Ahead Buffer (16 bytes)

LZWAH:

- 10 bit Code Size (1024 codes in dictionary)
- Flush Dictionary when full
- Adaptive Huffman encoding of LZW code words
- Rescale Huffman Tree at maximum weight of 32768

Three runs of each compression algorithm was made on the entire dataset on an unloaded Sun Sparc 5. Table 15 shown below lists the average time and average throughput for the various algorithms.

Algorithm:	Ave. Time (seconds)	Ave. Throughput (bytes/s)
Adaptive Arithmetic Compress	301.51	63,300
Adaptive Arithmetic Expand	479.59	34,700
Adaptive Huffman Compress	120.88	157,900
Adaptive Huffman Expand	137.47	122,200
LZAH Compress	449.78	42,400
LZAH Expand	129.80	125,300
LZWAH Compress	218.27	76,200
LZWAH Expand	223.12	74,500

Table 15

The results show that in terms of speed the adaptive Huffman algorithm had the best performance while the adaptive arithmetic algorithm had the worst. Adaptive arithmetic compression was roughly 2.5 time slower than its Huffman counterpart while expansion was about 4 times slower than Huffman. LZAH matched adaptive Huffman for expansion performance but was about 4 times slower in compression.

The following sections present a more detailed breakdown of the performance of the four algorithms.

5.1.1 Adaptive Huffman Compression

The primary data structure used for storage in adaptive Huffman compression is a binary tree.

```
struct tree
{
  int leaf[ 258 ];
  int next_free_node;
  struct node
  {
    unsigned int weight;
    int parent;
    int child_is_leaf;
    int child;
  } nodes[ 515 ];
} Tree;
```

This data structure requires 4638 bytes using 16 bit integers.

Table 16 shows a breakdown of the average amount of time spent in each procedure for adaptive Huffman compression. UpdateModel, OutputBits, and EncodeSymbol account for over 80% of the processing time.

Function:	Description:	Ave. Time	Ave. %
UpdateModel	Update Huffman tree	48.44	40.08%
OutputBits	Write a group of bits to disk	32.04	26.51%
EncodeSymbol	Huffman encode current symbol	21.28	17.60%
swap_nodes	Swap two nodes in Huffman tree	12.12	10.03%
CompressFile	Main Loop	5.38	4.45%
RebuildTree	Rescale node weights and rebuild tree	1.51	1.25%
add_new_node	Add a new node to Huffman tree	0.11	0.09%
InitializeTree	Initialize Huffman tree	0.00	0.00%
Total Time		120.88	100.00%

Table 16

Table 17 shows a breakdown of the average amount of time spent in each procedure for adaptive Huffman expansion. UpdateModel, InputBit, and DecodeSymbol account for over 80% of the processing time.

Function:	Description:	Ave. Time	Ave. %
UpdateModel	Update Huffman tree	49.80	36.23%
InputBit	Read a single bit from disk	36.68	26.68%
DecodeSymbol	Decode Huffman code	27.08	19.70%
swap_nodes	Swap two nodes in Huffman tree	12.43	9.04%
ExpandFile	Main Loop	8.28	6.02%
InputBits	Read a group of bits from disk	1.63	1.19%
RebuildTree	Rescale node weights and rebuild tree	1.41	1.03%
add_new_node	Add a new node to Huffman tree	0.15	0.11%
InitializeTree	Initialize Huffman tree	0.00	0.00%
Total Time		137.47	100.00%

Table 17

5.1.2 Adaptive Arithmetic Compression

The primary storage used by the adaptive arithmetic algorithm for both compression and expansion is an array of 258 integers. The algorithm works with 16-bit integers.

```
short int totals[ 258 ];
```

The first 256 elements of the array are used to store the count ranges for the 8 bit codes for the input symbols. The 257th element is used to store the count range for the END-

OF-STREAM symbol and the 258th element is used to store the total count. With 16 bit integers, this data structure requires 516 bytes of memory.

Table 18 shows a breakdown of the average amount of time spent in each procedure for adaptive arithmetic compression. AArithUpdateModel and AArithEncodeSymbol amount to 85% of the processing time used by adaptive arithmetic compression.

Function:	Description:	Ave. Time	Ave. %
AArithUpdateModel	Update symbol table	199.68	66.23%
AArithEncodeSymbol	Shift as many bits as possible to output	57.95	19.22%
OutputBit	Write a single bit to disk	31.21	10.35%
CompressFile	Main Loop	6.65	2.20%
AArithConvertIntToSymbol	Find low count, high count, and scale of symbol	4.46	1.48%
OutputBits	Write a group of bits to disk	1.55	0.52%
AArithInitModel	Initialize arithmetic symbol table	0.00	0.00%
AArithFlushEncoder	Shift out underflow bits and terminate arithmetic coding	0.00	0.00%
AArithInitEncoder	Set low count, high count, and underflow bits	0.00	0.00%
Total Time		301.51	100.00%

Table 18

Table 19 shows a breakdown of the average amount of time spent in each procedure for adaptive arithmetic expansion. AAirthUpdateModel, AArithConvertSymbolToInt and AArithRemoveSymbolFromStream account for nearly 90% of the processing time.

Function:	Description:	Ave. Time	Ave. %
AArithUpdateModel	Update symbol table	200.99	41.91%
AArithConvertSymbolToInt	Find low and high count for current symbol	166.53	34.72%
AArithRemoveSymbol FromStream	Convert arithmetic code to symbol, remove as many bits as possible from input buffer and read in new bits	58.72	12.24%
InputBit	Read a bit from disk	32.55	6.79%
ExpandFile	Main Loop	12.57	2.62%
AArithGetCurrentCount	Find count for current symbol	5.40	1.13%
AArithGetSymbolScale	Find scale	2.57	0.54%
AArithInitDecoder	Read first 16 bits to fill input buffer and set low, high count	0.26	0.05%
AArithInitModel	Initialize arithmetic symbol table	0.00	0.00%
Total Time		479.59	100.00%

Table 19

5.1.3 LZAH Compression

The LZAH algorithm requires the use of the data structures from the LZSS and adaptive Huffman algorithms. The binary tree used for the adaptive Huffman algorithm was shown in the previous section. Using 16 bit integers the binary tree required 4638 bytes. The LZAH algorithm also requires the text window from the LZSS algorithm and a binary tree for searching the window.

```
struct
{
  int parent;
  int smaller_child;
  int larger_child;
} tree[ 4097 ];

unsigned char window[ 4096 ];
```

With 16 bit integers and 8 bit characters the binary tree data structure requires 24582 bytes of storage and the window requires 4096 bytes of storage. The binary tree and window data structures are used by LZAH compression but only the window data structure is required for LZAH expansion. The binary tree for adaptive Huffman compression requires 4638 bytes. Therefore LZAH compression requires 33316 bytes of storage while LZAH expansion requires 8734 bytes of storage.

Table 20 shows a breakdown of the average amount of time spent in each procedure for LZAH compression. AddString and AHuffUpdateModel account for over 80% of the processing time.

Function:	Description:	Ave. Time	Ave. %
AddString	Add new nodes to LZ window binary search tree	262.26	58.31%
AHuffUpdateModel	Update Huffman tree	56.94	12.66%
OutputBits	Write a group of bits to disk	33.94	7.55%
DeleteString	Delete nodes from LZ window binary search tree	27.91	6.21%
AHuffEncodeSymbol	Huffman encode current symbol	25.99	5.78%
CompressFile	Main Loop	16.41	3.65%
HuffmanSwapNodes	Swap two nodes in Huffman tree	12.13	2.70%
LZAHReplaceNode	Called by DeleteString	12.35	2.75%
RebuildHuffmanTree	Rescale node weights and rebuild tree	1.40	0.31%
GetArguements	Parse command line arguments	0.19	0.04%
LZAHFindNextNode	Called by DeleteString	0.14	0.03%
HuffmanAddNewNode	Add a new node to Huffman tree	0.10	0.02%
InitHuffmanTree	Initialize Huffman tree	0.00	0.00%
Total Time		130.58	100.00%

Table 20

Table 21 shows a breakdown of the average amount of time spent in each procedure for LZAH expansion. AHuffUpdateModel, InputBit, and AHuffDecodeSymbol account for over 75% of the processing time.

Function:	Description:	Ave. Time	Ave. %
AHuffUpdateModel	Update Huffman tree	44.41	34.21%
InputBit	Read a single bit from disk	31.86	24.54%
AHuffDecodeSymbol	Decode Huffman code	24.03	18.52%
ExpandFile	Main Loop	12.87	9.92%
HuffmanSwapNodes	Swap two nodes in Huffman tree	11.39	8.78%
InputBits	Read a group of bits from disk	3.59	2.76%
RebuildHuffmanTree	Rescale node weights and rebuild tree	1.31	1.01%
InitHuffmanTree	Initialize Huffman tree	0.24	0.18%
HuffmanAddNewNode	Add a new node to Huffman tree	0.10	0.08%
GetArguments	Parse command line arguments	0.00	0.00%
Total Time		129.80	100.00%

Table 21

5.1.4 LZWAH Compression

The LZWAH algorithm requires the use of the data structures from the LZW and adaptive arithmetic algorithms. The array used for the adaptive arithmetic algorithm was shown in the previous section. Using 16 bit integers the array required 516 bytes. The LZWAH compression algorithm also requires the hash table used by the LZW algorithm and the LZWAH decompression algorithm requires the LZW decoding stack.

```
struct
{
  int code_value;
  int parent_code;
  char character;
} dictionary[ 1201 ];

char decode_stack[ 1201 ];
```

With 16 bit integers and 8 bit characters, the dictionary data structures would require 6005 bytes of storage and the decode stack would require 1201 bytes. The dictionary structure is used by both the compression and decompression algorithms while the decode stack is only used by the expansion algorithm.

An enlarged Huffman tree capable of handling 1024 input symbols is required by LZWAH. The data structure is shown below:

```
struct tree
```

```
{
  int leaf[ 1026 ];
  int next_free_node;
  struct node
  {
    unsigned int weight;
    int parent;
    int child_is_leaf;
    int child;
  } nodes[ 2051 ];
} Tree;
```

The Huffman tree requires 18462 bytes using 16 bit integers. In total the LZWAH compression algorithm requires 24 467 bytes and the decompression algorithm requires 25 668 bytes.

Table 22 shows a breakdown of the average amount of time spent in each procedure for LZWAH compression. AHuffUpdateModel and AHuffEncodeSymbol account for over 55% of the processing time.

Function:	Description:	Ave. Time	Ave. %
AHuffUpdateModel	Update Huffman tree	95.43	43.72%
OutputBit	Write a single bit to disk	31.44	14.41%
AHuffEncodeSymbol	Huffman encode current symbol	28.96	13.27%
HuffmanSwapNodes	Swap two nodes in Huffman tree	21.89	10.03%
RebuildHuffmanTree	Rescale node weights and rebuild tree	14.26	6.53%
find_child_node	Search LZW dictionary using hash table	13.87	6.36%
CompressFile	Main Loop	12.17	5.58%
HuffmanAddNewNode	Add a new node to Huffman tree	0.23	0.10%
InitHuffmanTree	Initialize Huffman tree	0.00	0.00%
Total Time		218.27	100.00%

Table 22

Table 23 shows a breakdown of the average amount of time spent in each procedure for LZWAH expansion. AHuffUpdateModel and AHuffDecodeSymbol account for over 55% of the processing time.

Function:	Description:	Ave. Time	Ave. %
AHuffUpdateModel	Update Huffman tree	97.68	43.73%
AHuffDecodeSymbol	Decode Huffman code	35.38	15.84%
InputBit	Read a bit from disk	31.17	13.95%
HuffmanSwapNodes	Swap two nodes in Huffman tree	20.64	9.24%
ExpandFile	Main Loop	18.86	8.44%
RebuildHuffmanTree	Rescale node weights and rebuild tree	14.03	6.28%
decode_string	Convert LZW code to symbol string	3.31	1.48%
InputBits	Read a group of bits from disk	1.80	0.81%
InitHuffmanTree	Initialize Huffman tree	0.26	0.12%
HuffmanAddNewNode	Add a new node to Huffman tree	0.22	0.10%
Total Time		223.36	100.00%

Table 23

5.2 Algorithm Implementation

The compression algorithms must be implemented in a real-time environment suitable for integration into the public telephone network. The primary performance goal would be the ability to encode and decode a 64kbit/s digital bit stream in real time. This translates into about 8000 bytes/s throughput. Table 24 summarizes the performance of the four compression algorithms.

Algorithm:	Average Bits per Symbol	Average Time (seconds)	Average Throughput (bytes/s)	SPARC binary (bytes)	Data Structures (bytes)
Adaptive Arithmetic Compress	7.03	301.51	63,300	11 694	516
Adaptive Arithmetic Expand		479.59	34,700	11 115	516
Adaptive Huffman Compress	7.11	120.88	157,900	15 643	4638
Adaptive Huffman Expand		137.47	122,200	14 991	4638
LZAH Compress	6.88	449.78	42,400	19 237	33 316
LZAH Expand		129.80	125,300	18 578	8 734
LZWAH Compress	7.05	218.27	76,200	17 990	24 467
LZWAH Expand		223.12	74,500	17 346	25 668

Table 24

The results obtained from the GNU profiler indicate that adaptive Huffman gives the best performance in terms of speed by at least a factor of two compared to adaptive arithmetic, LZAH, and LZWAH compression. The disadvantage of adaptive Huffman compression is a lower compression ratio than the other three compression algorithms. However this ratio is usually only 5% to 10% lower than the compression ratio of the other algorithms.

LZAH gives the best performance in terms of compression. The disadvantage is the asymmetry between the compressor and decompressor. The compressor requires about three times the computational power of the decompressor.

Adaptive Huffman compression has a throughput of about 150,000 bytes/s and expansion has a throughput of 120,000 bytes/s on an 85MHz 32 bit SPARC 5 processor. The expansion rate of 120,000 bytes/s is about 15 times faster than the 8000 bytes/s required for any systems to be implemented in the public telephone network. The following estimates can be made assuming a 16 bit RISC processor core similar to the SPARC processor can be obtained:

- Assume a 85MHz 32 bit RISC processor has a throughput of 120,000 bytes/s.
- An equivalent 85MHz 16 bit RISC processor should have half the performance at about 60,000 bytes/s.
- An equivalent 11.3MHz 16 bit RISC processor should have a performance of about 8,000 bytes/s.

Therefore a 12MHz 16 bit RISC processor should be sufficient to run GCC compiled code for either adaptive Huffman compression or expansion. Other options available to reduce cost include:

- Optimization the adaptive Huffman algorithm using assembly language to allow the use of a slower 16 bit RISC processor.
- Since the adaptive Huffman algorithm spends about 40% of its time in the procedure UpdateModel, this procedure could be moved into a hardware co-processor. This would allow a slower 16 bit RISC processor to be used.
- Implementation the entire adaptive Huffman algorithm in custom VLSI hardware.

Table 24 also shows the memory requirements for the SPARC binaries and data structures used by the various compression algorithms. The adaptive Huffman algorithm requires about 20 kbytes of memory for the instructions and data based on the SPARC processor. Using a 16 bit RISC processor would require less memory for the instructions since they are 16 bits wide rather than 32 bits.

From this information, a minimum implementation of the adaptive Huffman compression and expansion algorithms would require a 16 bit RISC processor running at 12MHz with 20kbytes of RAM. Since the adaptive Huffman algorithm spends approximately 40% of its time in the UpdateModel procedure, moving this procedure into a hardware co-processor could reduce the workload of the processor by 40%. This would allow the use of a slower clock speed such as 8MHz. Coding the Huffman algorithm using assembly lanuage would reduce the memory requirements and improve performance as well. It should be possible to fit the code and data into 16kbytes of memory.

For any implementation in the public telephone network, adaptive Huffman compression would prove to be the best solution in terms of simplicity, speed, and cost. A larger issue

not discussed in this thesis is how to integrate a variable bit-rate compression system into a fixed bit-rate network such as the public telephone system. This may prove to be more complicated than implementing the adaptive Huffman compressor.

Chapter 6 : Conclusions

The vector quantization systems developed by [Ande85], [Rich88], and [Tran87] are examples of what fixed bit-rate voiceband data signal compression systems might look like. Although these systems were designed to handle modems up to 4800 bits/s they could be adapted to handle high-speed modems such as v.34 28.8 kbits/s. These compression systems would be able to handle most types of modem signals without specific knowledge about the exact type of modulation scheme. They may introduce a small amount of distortion into the modem signals but this is not likely to have a significant impact on performance. The compression ratio is inferior to demodulating the modem signals, and the compression system is likely to be complex.

The advantage of lossless compressors is that they reproduce the input signal exactly. This is important for high speed modem signals which are susceptible to the slightest distortions. The main disadvantages are that lossless compressors have poor compression ratios and variable bit-rate outputs. This makes them unsuitable for the circuit-switched telephone network, which uses primarily fixed bit-rate channels.

The ultimate goal of any voiceband data signal compression system would be to demodulate the data signal and transmit the information digitally. There are commercial products capable of doing this. The disadvantage of these systems is that they are limited to demodulating the modem signal types supported by the manufacturer. In the case of the InvsiLink system produced by DSP Software Engineering, it only supports v.22bis, v.32bis, and G3 fax modems at present.

One factor not considered above in the use of lossy or lossless voiceband data signal compression schemes is the cost of research, development and implementation. Most of these compression systems are complex, requiring considerable development resources. The benefits of these systems may therefore be outweighed by their cost. There are however a few specialized areas where such systems would be beneficial, such as when bandwidth is at a premium. In general, the widespread use of any voiceband data signal compression systems within the public switched telephone network is unwarranted. Lossy compression systems would be better applied at the source rather than at the transmission level and variable bit-rate output lossless compression systems would be difficult to integrate into a fixed bit-rate telephone network.

Although not discussed in this thesis, another more pressing problem is appearing in the telephone network. With the recent interest in the Internet and telecommuting, there has been a substantial increase in the use of modems and facsimile machines. The telephone network was originally designed to handle voice calls. The assumption was that average calls would of a short duration, on the order of minutes. The actual number of connections at any given time would be a small percentage of the total number of

telephone sets installed. Therefore a switch with a small number of connections could be used to route a large number of telephone sets. The problem introduced with modems and fax machines is that they have completely different access patterns compared to humans making voice calls. A typical modem user could connect to a Internet service provider or corporate server for many hours at a time. Busy fax machines could be receiving and making calls every minute. This type of usage places a load on the public telephone network switches that they were not designed for. At best, this would cause congestion on the network and new calls would not be connected. At worst, the load could actually cause telephone network switches to crash. Such performance is unacceptable.

6.1 Recommendations

Further research into the area of voiceband data compression, while of academic interest, will exhibit minimal practical benefit. The optimal solution to the wasted bandwidth problem is to upgrade the telephone network to digital end-to-end, by employing the Integrated Services Digital Network (ISDN) or other standards available in the future. If bandwidth is at a premium then a commercial demodulation-remodulation system such as InvisLink produced by DSP Software Engineering could be considered. The use of lossless compressors for voiceband data signals is technically feasible but not economically justifiable. The time and resources required to develop and deploy such a system would be better spent upgrading the telephone network to ISDN.

6.2 Alternative Research Directions

Portions of this research may be of more benefit when applied in reverse to increase modem data rates. For example, J. Foster and S. Harris present a novel method of high speed modem design using vector quantization [FoHa91].

In terms of lossless compression, it might be interesting to examine higher order modeling such as an order-1 or order-2 model for the statistical compressors. The LZSS compressor encoded as much as 50% of the input symbols as two symbol match pairs. This would suggest that a higher order statistical compressor would be able to achieve substantially more efficient coding of the input symbols.

Bibliography

[Ande85] D. O. Anderton, "Vector quantization of voiceband data signals," Ph.D. dissertation, Univ. Utah, Salt Lake City, Dec. 1985.

[BCW90] T. C. Bell, J. G. Cleary, I. H. Witten, *Text Compression*, Prentice Hall, 1990

[Bell86] T. C. Bell, "Better OPM/L text compression," IEEE Transaction on Communications, vol. 34, no. 12, December 1986, pp. 1176-1182.

[Bren87] R. P. Brent, "A linear algorithm for data compression," *Australian Computer Journal*, vol. 19, no. 2, pp. 64-68.

[DiCo93] S. Dimolitsas, F. L. Corcoran "Non-voice performance of the 16kbit/s LD-CELP algorithm," *Speech Communication*, vol. 12, Iss: 2, pp. 145-150, June 1993.

[Dimo93] S. Dimolitsas, "Characterization of low-rate digital voice coder performance with non-voice signals," *Speech Communication*, vol. 12, Iss: 2, pp. 135-144, June 1993.

[FiGr89] E. R. Fiala, D. H. Greene, "Data compression with finite windows," *Communications of the ACM*, vol. 32, no. 4, April 1989, pp. 490-505.

[FoHa91] J. Foster and S. Harris, "16-kbps Modem Design using Vector Quantization," *IEEE SouthEastCon '91,* vol. 2, pp. 1036-1039, April 1991.

[Huff52] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, September 1952, pp. 1098-1101.

[ITU90] ITU-T Rec. G.726 (1990), "40, 32, 24, 16 kbit/s Adaptive Differential Pulse Code Modulation (ADPCM)," Geneva, p. 56.

[JaNo84] N. S. Jayant and P. Noll, "Digital Coding of Waveforms, Principles and Applications to Speech and Video," Prentice Hall, Englewood Cliffs, New Jersey, 1984.

[LKM90] D. Lin, S. D. Kurtz, and B. M. McCarthy, "Data Compression of Voiceband Modem Signals," *40th IEEE Vehicular Technology Conference*, pp. 323-325, May 1990.

[Mart90] J. Martin, "Telecommunications and the computer," 3rd ed., Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1990

[MZO75] P. J. May, C. J. Zarcone, and K. Ozone, "Voiceband Data Modem Performance Over Companded Delta Modulation Channels," *IEEE Trans. Commun.*, vol. 23, pp. 1495-1502, Dec. 1975.

[Nels92] M. Nelson, *The Data Compression Book*, M&T Publishing, Inc., 1992.

[ONSt72] J. B. O'Neal Jr., and R. W. Stroh, "Differential PCM for speech and data signals," *IEEE Trans. Commun.*, vol. 20, pp. 900-912, Oct. 1972.

[ONea74] J. B. O'Neal Jr., "Delta Modulation of Data Signals," *IEEE Trans. Commun.*, vol. 22, pp. 334-339, Mar. 1974.

[OpSc89] A. J. Oppenheim and R. W. Schafer, "Discrete-time signal processing," Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[Rich88] D. S. Richards, "Baseband template - phase vector quantization of voiceband data signals," Ph.D. dissertation, Univ. Utah, Salt Lake City, Dec. 1988.

[Satc94] S. Satchell, "The Other Modem," *PC Magazine*, vol.13 no.15, pp. 286-287, Sept. 13, 1994.

[Shan48] C. E. Shannon, "A mathematical theory of communication," *Bell Systems Technical Journal*, vol. 27, July 1948, pp. 398-403.

[ShHe94] T. Shamoon and C. Heegard, "A Rapidly Adaptive Lossless Compression Algorithm for High Fidelity Audio Coding," *Proceeding DCC '94 Data Compression Conference*, Snowbird Utah, 1994, pp. 430-439

[Strem90] F. G. Stremler, "Introduction to Communication Systems," 3rd Edition, Addison-Wesley, Don Mills, Ontario, 1990.

[StSz82] J. A. Storer, T. G. Szymanski, "Data compression via textual substitution," *Journal of the ACM*, vol. 29, no. 4, October 1982, pp. 928-951.

[Tisc87] P. Tischer, "A modified Lempel-Ziv-Welch data compression scheme," Australian Computer Science Communications, vol. 9, no. 1, pp. 262-272.

[Tran87] T. D. Tran, "Bauded signals compression using the baseband residual vector quantization algorithm," Ph.D. dissertation, Univ. Utah, Salt Lake City, Dec. 1987.

[Vaug95] S. J. Vaughan-Nichols, "Phone Lines Stymie v.34 Modems," Byte Magazine, vol. 20, no. 11, November 1995, p. 40.

[Welc84] T. Welch, "A Technique for High-Performance Compression Standard," *IEEE Computer*, vol. 17, no. 6, June 1984, pp. 8-19.

[WNC87] I. H. Witten, R. M. Neal, J. G. Cleary, "Arithmetic Coding for Data Compression," *Communications of the ACM*, vol. 30, no. 6, June 1987, pp. 520-540.

[ZiLe77] J. Ziv, A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, May 1977, pp. 337-343.

[ZiLe78] J. Ziv, A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, September 1978, pp. 530-536.