2005

AZIP, audio compression system: Research on audio compression, comparison of psychoacoustic principles and genetic algorithms

Howard Chen

AZIP, AUDIO COMPRESSION SYSTEM: RESEARCH ON AUDIO COMPRESSION, COMPARISON OF PSYCHOACOUSTIC PRINCIPLES AND GENETIC ALGORITHMS

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Howard Chen

March 2005

AZIP, AUDIO COMPRESSION SYSTEM: RESEARCH ON AUDIO COMPRESSION, COMPARISON OF PSYCHOACOUSTIC PRINCIPLES AND GENETIC ALGORITHMS

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Howard Chen

March 2005

Approved by:

[redacted]

Dr. Tong Lai Yu, Chair, Computer Science

3/10/05
Date

[redacted]

Dr. Kay Zemoudeh

[redacted]

Dr. Keith Schubert

ABSTRACT

The purpose of this project is to investigate the differences between psychoacoustic principles and genetic algorithms (GA). These two theories will be discussed separately. The review will also compare the compression ratio and the quality of the decompressed files decoded by these two methods.

The report will thoroughly discuss the reason that psychoacoustic principles produce small compressed files without affecting the quality of the sound. Two important topics, absolute threshold of hearing and masking effect, will be discussed in the report.

Unlike psychoacoustic principles, GA uses a different approach to achieve high compression ratio. In the review, topics related to GA will be discussed. These topics will include the concept, training procedure, and the performance of GA.

A computer program called Azip has been developed and is included for the use of researchers who are interested in this topic. In the review, the design and implementation of Azip will be discussed. Several experiments have been done using Azip. The result of experiments is discussed in the report as well.

The review also compares the performance of Azip with that of Ogg and MPEG Layer3 (MP3), so the readers can have an understanding on their differences.

ACKNOWLEDGMENTS

Firstly, I would like to thank my supervisor, Professor Tong Lai Yu for his supervision and his support. He provided lots of valuable information that extended my understanding in audio compression and eventually improved the quality of the project.

I would also like to thank my two-committee members, Dr. Kay Zemoudeh and Dr. Keith Schubert, for their suggestions and help.

I would like to thank Dr. Jenq-Shyang Pan and Yi-Chih Hsin for introducing and discussing genetic algorithms. They also gave me lots of help in solving many problems and debugging many programs.

I would like to extend my thanks to many other colleagues and friends helped me during my master study. In no particular orders, thanks to: Chun-Hsien Wang, John Ho, Jacob Lee, Chao-Kang Lee, Chuan-Ting Chou, Tang-Li Chen, Steven Chih, and Dennis Hsieh.

Finally, I wish to thank to my parent and my sister for their encouragement and support.

The support of the National Science Foundation under award 9810708 is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Purpose of the Project

The purpose of this project is to investigate the theories and the performance of psychoacoustic principles and genetic algorithms. Psychoacoustic principles are theories specifically designed and applied to audio compression. It is noted that sound can be masked by other sounds and the resulting sound can't be detected by human ear. Because of this, psychoacoustic principles focus on removing the irrelevant sound and thus achieving compression. The method has been adopted by most modern audio compression systems, including MPEG Layer3, AAC and Ogg. Unlike psychoacoustic principles that detect and remove irrelevant sound, genetic algorithms analyze a lot of audio files to return a string of numbers. These numbers represent the subbands that are least likely to influence the quality of music if they are removed. The programs that use genetic algorithm simply remove the subbands in the solution. In this project, the genetic algorithm will be adapted to audio compression. The result will be compared with the one produced by psychoacoustic

principles. The performance of MPEG Layer3 and Ogg will be compared with that of Azip.

Signal-to-Noise Ratio (SNR) will be used as the standard for comparing the quality of the decompressed files produced by the decoders based on psychoacoustic principles and genetic algorithm. Although decompressed files with larger SNR value should have a better quality in theory, it is not necessary the case in reality because human ear is not sensitive enough to detect all differences. To solve the problem, subjective testing will be used to find the file that has the better quality. The subjects selected for sound quality test can be ranged from naïve listeners, trained listeners, to listeners who are gifted in all areas of auditory perception, also called "golden ear". During the subjective testing, listeners hear pairs of music files and give a grade for each pair. Further detail about subjective testing will be discussed in 5.2.2.

1.2 The Scope of the project

The audio compression system contains the following three components

1. The coder and decoder using psychoacoustic principles

2. The coder and decoder using genetic algorithm

3. Signal-to-Noise Ratio used for determining the quality of the decompressed files produced by the decoder.

The coder could compress one or more .wav files to .csusb files using the method specified by the users. The decoder decompresses .csusb files to .wav files. The users can easily change the method they want to compress the .wav files by modifying the configuration. The processes of compression and decompression are illustrated in figure 1.

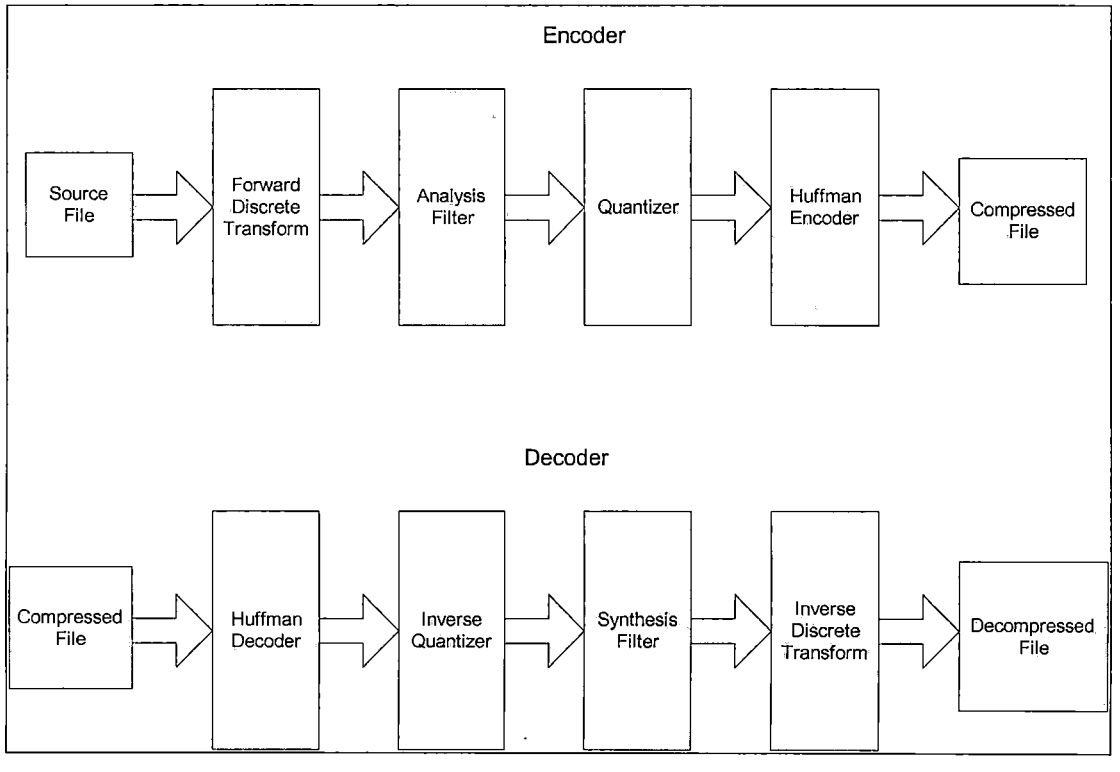


Figure 1. Basic Audio Compression and Decompression Process

Every compression program contains a coder and a decoder. Figure 1 shows a basic flowing process in a typical modern audio compression program. The purpose of the compression process is removing all undetectable signals, reducing redundancy of raw data and storing the digital representation in a new file. In order to reduce the size of the compressed files, Azip first analyzes the input file by applying discrete cosine transform (DCT) or modified discrete cosine transform (MDCT). In the analysis filter, analysis filter determines the undetectable sound. This includes the sound that is masked by other signals because of masking effect. Besides this, the sound that does not have enough energy to be detected by human ear would also be identified.

Quantizer removes the undetectable sound in the data. Lossless algorithm is applied next to remove redundancy in the remaining data. In the project, Huffman encoder is used in the lossless compression process. The final data is stored in a .csusb file.

The decoder transforms the data in .csusb file from time-frequency representation back to the time-domain representation which can be played in Windows Media Player or other music playing programs. Decoder also contains three sections, Huffman decoder, inverse quantizer and

synthesis filters. The goal for each section is to produce the information sent to the correspond section in encoder. Huffman decoder reverses the effect of Huffman encoder and inverse quantizer de-quantizes the data sent from Huffman decoder. The output of the inverse quantizer is sent to synthesis filters and inverse discrete transform at which the time-domain representation will be reconstructed.

1.3 Organization of Chapters

Chapter two will focus on Psychoacoustic principles. This will include some physics phenomena that are used in audio compression. Chapter three covers on the concept of genetic algorithm. The processes used to find a solution will be described in detail as well.

The architecture of the project is provided in chapter four. This chapter describes each component of the project and the relationships among them. User interface will be briefly discussed in chapter four as well. The chapter also introduces the implementation of the project

Chapter five compares the result of genetic algorithm and psychoacoustic model. The computation and complexity of modified discrete cosine transform (MDCT) and discrete cosine transform (DCT) will also be discussed. Furthermore, the result of psychoacoustic model and

genetic algorithms will be compared in the chapter. This will include compression ratio, SNR comparison and the result of subjective testing.

Chapter six lists of improvement and concludes this paper.

CHAPTER TWO

PSYCHOACOUSTIC PRINCIPLES

2.1 Introduction

Psychoacoustic principles achieve compression by detecting the signal information that is not detectable by human audio system. In order to detect the irrelevant signal, human auditory perception and particularly the time-frequency analysis capabilities of the inner ear have been modeled and researched [5]. Modern audio compression software applications use psychoacoustic principles to exploit the irrelevant signals. By effectively eliminating the irrelevant information, the storage requirement can be greatly reduced.

Two major techniques used by psychoacoustic principles are absolute threshold of hearing and masking effect. Absolute threshold of hearing emphasizes the limitation of human ear in quiet condition. Masking effect focuses on the phenomena that a signal masks or is masked by another signal. Absolute threshold of hearing will be discussed in 2.2 and masking effect would be discussed in 2.3.

2.2 Absolute Threshold of Hearing

Sound is characterized by its energy in physics. Absolute threshold of hearing characterizes the amount of energy needed in a pure tone such that it can be detected by a listener in a noiseless environment [5]. Absolute threshold is expressed in db sound pressure level (dB SPL). Figure 2 shows the relationship between absolute threshold and frequency.

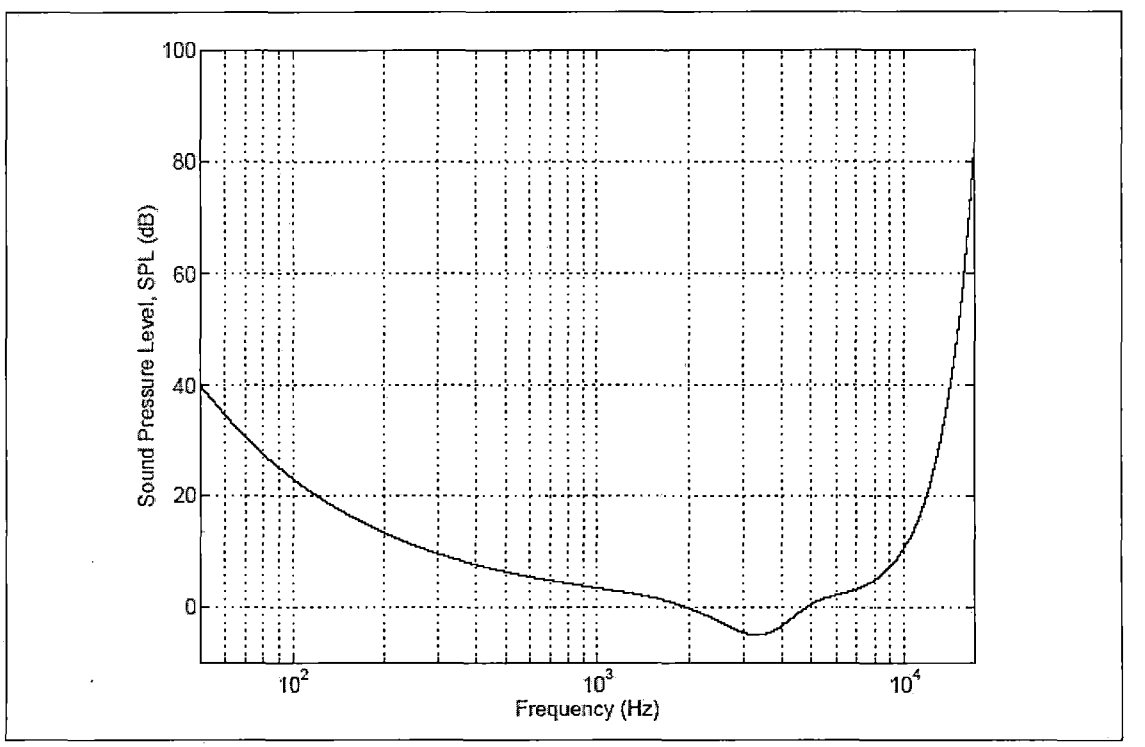


Figure 2. The Absolute Threshold of Hearing Under Quiet Condition

The non-linear function shows the sound pressure level (SPL) required at each frequency for average

listeners to detect the signal. From the point of view of audio compression, this curve represents the maximum energy level for coding distortion introduced in the frequency domain [5]. In other words, if we plot the point that represents a sound signal in the graph and the point is under the curve, the signal can be removed because it is too quiet for anyone to hear. In contrast, the signal that is higher than the curve must be reserved or listeners will detect the differences. From the graph, we can see that human ears are most sensitive to the sound near 4kHz. A 4kHz sound signal with low energy level can still be easily detected by most people. However, a high frequency sound with much higher energy level is undetectable for most people.

We can conclude that whether or not a sound signal can be heard depends on its two properties, frequency and the amount of energy it carries. Different sound signals with same energy level can be loud, quiet or undetectable by human depending on its frequency. By applying this to audio compression, we can remove signals that nobody can hear under quiet condition and obtain a better compression effect.

2.3 Masking Effect

The absolute threshold of hearing determines the amount of energy required for people to hear the sound with certain frequency. However, music is created by sound signals each with different frequency and sound pressure level. Each sound signal can influence the neighbor sound signals as well. Masking refers to one sound that is inaudible because of the presence of another sound.

The masking effect allows us to further remove the sound signals that are masked. By removing the signal, we reduce the size of a music file and achieve a better compression result. Masking can be further divided to simultaneous masking and nonsimultaneous masking.

2.3.1 Simultaneous Masking

When two or more noises exist simultaneously, the sound with higher SPL will mask off the one with lower SPL. Simultaneous masking happens when the presence of a strong masker creates sufficient strength to block detection of a weaker signal [5]. The phenomena can be further divided to tone-masking-noise (TMN), noise-masking-tone (NMT), and noise-masking-noise (NMN).

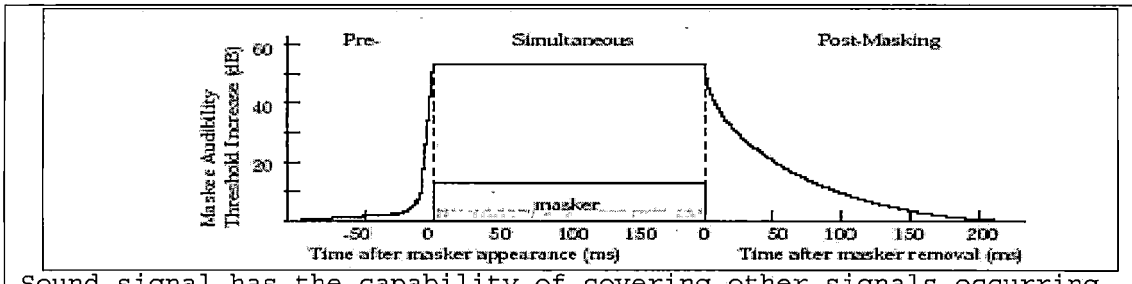
When a tone and a noise within the same critical band exist simultaneously, the minimum signal-to-mask ratio (SMR) determines whether we only hear the tone, the noise

or both of them. SMR is the difference between the SPL of masking signal and the SPL of masked signal.

Tone-masking-noise occurs when tone and noise are within the same critical band, and the SMR ratio is between 21-28 dB. In the case the pure tone is the masker and covers the noise (maskee). Most people are able to detect the tone even when noise exists. In contrast, noise-masking-tone (NMT) happens when the SMR is between -5 and 5 dB. The tone would be covered by the noise in NMT. Under this situation, nobody is able to hear the tone because it is covered by the surrounding noise.

2.3.2 Non-Simultaneous Masking

Non-simultaneous masking refers to the phenomena that a masker can cover the signal that is right before or after it. The fact that a masker can cover the sound prior to masker onset is called pre-masking. Figure 3 illustrates the property of non-simultaneous masking.



Sound signal has the capability of covering other signals occurring very close to it. The length of post-masking is much longer than the length of pre-masking.

Figure 3. Non-Simultaneous Masking

Post-masking, also called forward masking, refers to the situation that a masker can cover another sound even after the masker is removed. Unlike post-masking, which lasts more than 100ms, pre-masking, also called backward masking, usually only lasts for a few ms. From figure 3, we can see that sound signals do not only cover the sound that occurs simultaneously, but under some conditions, it has the ability to make all sound that occurs very close to the masker undetectable. Taking advantage of this phenomenon, one can improve audio compression ratio.

By applying absolute threshold of hearing and masking effect, psychoacoustic principles can accurately detect the signal that can not be heard. The information that represents the irrelevant sound can be removed but the quality of the music file can be kept.

CHAPTER THREE

GENETIC ALGORITHMS

3.1 Introduction

Genetic algorithms are a series of methods which solve problems by modeling the process of Darwinian evolution [16]. Psychoacoustic principle determines and removes the irrelevant signal of an individual music file. In contrast, genetic algorithms analyze a large number of music files and determine the chunks that are most likely to contain irrelevant signal. The combination of these chunks is called the solution. In the training process, sequences of processes are repeatedly executed until the requirements are met. The outcome of the training process is the solution that will be used in the compressing process. Unlike psychoacoustic principle, genetic algorithms do not analyze music files anymore. This method removes chunks that are listed in the solution regardless of music type and length. Because of this, the solution encoded in the program plays an important role and directly influences the performance of genetic algorithms.

3.2 Training Processes

In the training processes, a large number of solutions are generated by the genetic algorithms first.

Each of these solutions is called a chromosome. Each chromosome is made of a string of different values. Each value is a gene. The value and the order of the genes in a chromosome determine the feature of the chromosome.

The genes of the first generation chromosomes are selected from random. During the training processes, two chromosomes crossover and create chromosomes of the next generation. The genes of the child will be determined by the genes of the parents. The newly created child will have a chance to mutate. When a chromosome mutates, the value of one or more genes in the chromosome will be changed to a randomly selected number. The processes of crossover and mutation may create new chromosomes that perform better than their parents. In order to keep the chromosomes that suit our requirements mostly, an evaluation is executed. Each chromosome would be given a score in the evaluation process. The score, also called fitness value, determines which chromosomes are selected as the parents for the chromosomes of the next generation. The processes will be repeated, and chromosomes with higher fitness value will be generated. The cycle continues until a suitable chromosome is found. The process of genetic algorithms is as follows.

Step 1. Initialization: Create chromosomes by randomly selecting the genes.

Step 2. Evaluation: Evaluate the fitness of each chromosome.

Step 3. Selection: Select the chromosomes with better fitness as the candidates for crossover and mutation operations.

Step 4. Crossover: Two candidates are selected as parents to produce new chromosomes of next generation.

Step 5. Mutation: Mutation is randomly applied to the genes of new chromosomes.

Step 6. Repeat steps 2 to 5 until a suitable chromosome is found.

The training processes are illustrated in figure 4.

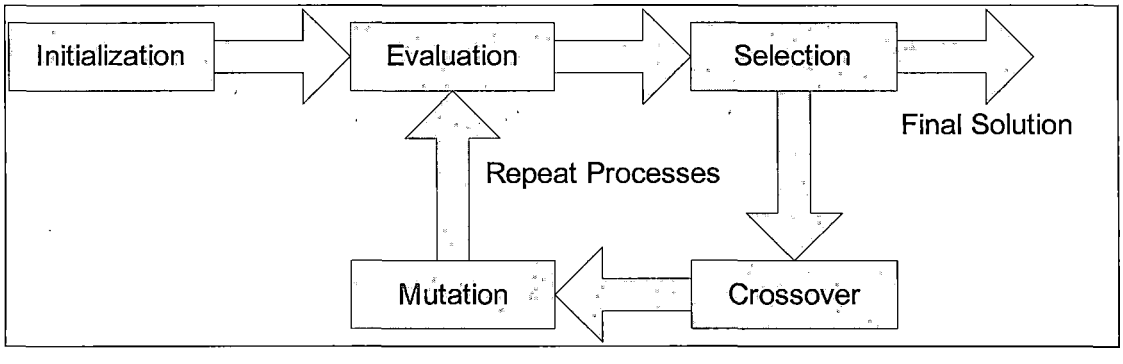


Figure 4. The Training Procedures of Genetic Algorithms

3.2.1 Initialization

A set of chromosomes is generated randomly. Each chromosome contains fixed number of genes.

3.2.2 Evaluation

Evaluation is the process of determining the fitness of chromosomes. In Azip audio compression system, Signal-to-Noise Ratio (SNR) is used to determine the fitness. Chromosomes are sorted according to their fitness.

3.2.3 Selection

The chromosomes with higher fitness value would be selected as the candidates for crossover and mutation. Two common selection methods are roulette wheel selection and tournament-based selection. In order to generate better chromosomes, both methods give chromosomes with higher fitness value better chances to be selected. The chromosomes with low fitness value may contain good genes, therefore, both methods reserve a small chance for chromosomes with low fitness value.

3.2.4 Crossover

The crossover operation is one of the most important operations in genetic algorithm. The basic idea is to generate new chromosomes by exchanging the segments between pairs of chromosomes. Some crossover operations will be illustrated here.

3.2.4.1 One-Point Crossover. The one-point crossover is a simple and effective technique. The first step of the

technique is to select the crossover point. Genes up to and including the crossover point are copied from one parent. All other genes are copied from the second parent. Assume the chromosomes of two parents are as follow

Parent1: 2 30 45 56 77 82 90 95

Parent2: 12 22 39 48 50 64 83 92

If position 4 is selected to be the crossover point, then the following two children will be generated.

Child1: 2 30 45 56 | 50 64 83 92

Child2: 12 22 39 48 | 77 82 90 95

The first child has the first four genes from parent 1 and others from parent 2. The second child has the first four genes from parent 2 and others from parent 1.

3.2.4.2 Two-Point Crossover. The procedure is similar to one-point crossover. The only difference is instead of one crossover point, two points are selected and genes between these two points are swapped. With the same parents above and position 2 and 5 as the crossover points, the following two children will be generated

Child1: 2 | 22 39 48 50 | 82 90 95

Child2: 12 | 30 45 56 77 | 64 83 92

3.2.4.3 n-Point Crossover. The procedure is similar to two-point crossover except that n crossover points are selected. Genes between odd and even crossover points are

swapped. All other genes remain the same. With the same parents above and position 2, 5, 6, and 7 as the crossover points, the following two children will be generated.

Child1: 2 30 | 39 48 50 | 82 | 83 | 95

Child2: 12 22 | 45 56 77 | 64 | 90 | 92

3.2.4.4 Uniform Crossover. In uniform crossover [14], each gene is copied from a parent based on a random flip. In other words, for each gene, the probability that the gene is copied from parent 1 is equal to the probability that it is copied from parent 2.

Crossover allows two chromosomes to generate two new chromosomes. In audio compression, not all chromosome generated by the crossover is valid and effective. The most important restriction is that the value of every gene must be unique. If two or more genes contain the same value in one chromosome, the chromosome is invalid. However, in the view of audio compression, the order of the genes is not meaningful. For example, a chromosome with value {1, 2, 3, 4, 5} is the same as a chromosome with value {5, 3, 2, 4, 1}.

3.2.5 Mutation

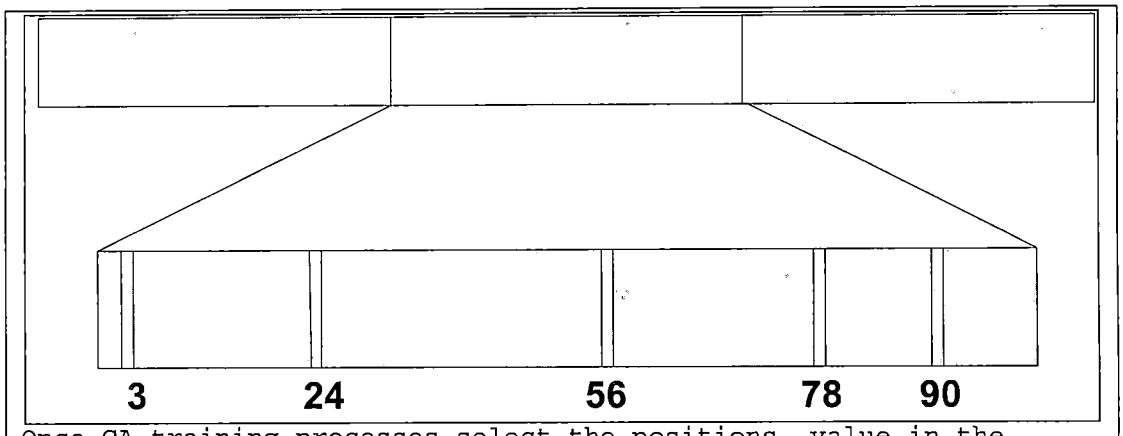
Although selection and crossover effectively search and generate new chromosomes, they may miss some potentially useful genes. In some cases, some useful genes

may not even be selected in the initial step. In order to produce a better solution, mutation is applied. The basic idea of mutation is to randomly select a number as crossover point. The value of the gene in the position would be replaced by any valid random number that does not exist in the chromosome.

3.3 Applying Genetic Algorithms on Audio Compression

To model the process of Darwinian evolution, a music file is first cut to several sections with equal length. In the case that the last chunk is not long enough to meet the required length, a number of zeros are added to extend the length of the last chunk.

As illustrated in figure 5, two music files are allowed to have different number of sections. However, the length of each section must be the same regardless of the length of music files.



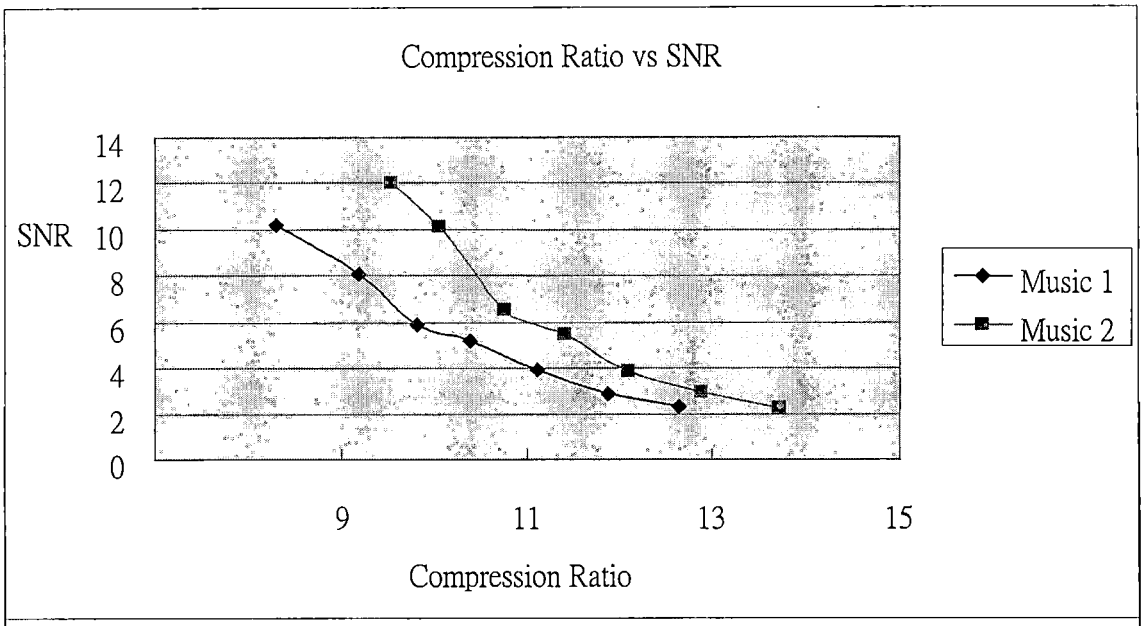
Once GA training processes select the positions, value in the position will be changed to 0. In figure 6, chromosome with {3, 24, 56, 78, 90} is selected as solution in the training processes. During the compressing process, positions 3, 24, 56, 78, 90 will all change their values to 0. (same applies for the first and third chunk)

Figure 6. Compressing Process of Genetic Algorithm

The length of a section directly influences the length of the solution. The length of the solution must be between zero and the length of a section. Genetic algorithm achieves compression effect by changing the chunk whose position is in the solution. The values of these chunks will all be changed to zero. The action increases the redundancy and creates better compression effect during the Huffman coding.

If the length of the solution is zero, no compression will take place and the decompressed output file is identical with the original one. As the length of the solution increases, the compression ratio increases but the quality of the decompressed file drops. Figure 7

illustrates this by comparing the compression ratio with the SNR.



Genetic algorithms are applied to two different music. The length of the solution increases from 10% to 70% of the length of the whole music files. The increase of the solution length causes the compressing ratio to increase but the SNR decreases.

Figure 7. Compression Ratio versus Signal-to-Noise Ratio in Genetic Algorithms

3.4 Problem of Genetic Algorithm and Conclusion

Genetic algorithm is a simple technique to compress an audio file, but it also has a problem. The main problem of the method is that the final solution produced is a general solution. Genetic algorithm only uses one solution to compress all kind of music files. Consequently, the differences between some decompressed files and the source files are significant. In order to generate the best result for a music file, the training and evaluation

process must be executed every time. This will be time-consuming and inefficient.

One way to create a better result using genetic algorithm is to produce multiple solutions. Each solution is specifically designed for a kind of music, for instance, classical music. The performance of the genetic algorithm for this kind of music will be improved.

CHAPTER FOUR

PROJECT DESIGN AND IMPLEMENTATION

4.1 Overall Description

4.1.1 Project Functions

Azip audio compression system is an application which allows the users to compress/decompress wave files. During the compressing process, a file with ".csusb" suffix will be generated. When a .csusb file is decompressed, the application will generate a .wav file. Besides compressing and decompressing files, users can also easily get the file format of a .wav file. By using command "snr", users can find the signal-to-noise ratio of two .wav files. The usage of these commands will be discussed in 4.1.3.

4.1.2 Hardware Interface

Azip audio compression system is tested under Cygwin. Since Cygwin is a Linux-like environment run under Windows, Azip is able to run under Linux operation system.

4.1.3 User Interface

4.1.3.1 Azip Commands. Azip is a text form application. Table 1 shows all legal commands and their functions

Table 1. Legal Commands in Azip Audio Compression System

azip -c [list of wave files]	Compress a list of wave files
azip -d [list of .csusb files]	Decompress a list of .csusb files
azip -i [list of wave files]	Print the format of one or multiple wave files
snr file1.wav file2.wav	Calculate the SNR of file1 and file2

For the users use Cygwin system, please add "./" before each commands.

4.1.3.2 Configuration Modification. In order to modify the configuration values, one must open a file called "codec_config.h". The content of the file looks like the following figure.

```
#ifndef CODEC_CONFIG
#define CODEC_CONFIG


#define QFACTOR 0.2
#define QUANTIZATION
#define DYNAMIC_QFACTOR
//#define DEBUG
//#define USE_MDCT
#define USE_DCT
#define FREQ_MASK
//#define MY_MASK

#endif
```

Figure 8. File Codec_config.h

QFACTOR controls the compression rate. By setting it lower, compression ratio will rise but the quality of the music will drop. By adding or removing "//" sign before each line, users can control the way the program compress the file. When "//" is added before a line of code, this line of code is inactive. Figure 8 shows the setting for the combination of DCT and psychoacoustic principle. Once codec_config.h is modified and saved, users must type "make clean all" under Cygwin or Unix to make the new setting effective.

4.2 Project Functions Design and Implementation

The section will discuss some important files used in Azip audio compression system. The files in Azip audio compression system are summarized in table 2.

Table 2. Files in Azip Audio Compression System

File Name	Function
audio.c	.wav file handling, DCT and MDCT
decode.c	Decompressing .csusb files to .wav files
encode.c	Compressing .wav file to .csusb files.
Freqmask.c	Using psychoacoustic principles to compress data.
ga.c	Executing training processes.
huffman.c	Creating Huffman tree and Huffman table. Generate the compressing code used in Quantization process
snr.c	Calculate the signal-to-noise ratio of two different .wav files

4.2.1 Filter Implementation

Two kinds of filters, DCT and MDCT, are implemented in Azip audio compression system. They are coded under audio.c files. The functions implement DCT and MDCT are listed in table 3.

Table 3. Critical Functions Used in Discrete Cosine Transform and Modified Discrete Cosine Transform

Function Name	Purpose
init_dct	Initializes tables required in DCT
init_mdct	Initializes tables required in MDCT
Dct	Main DCT program
Mdct	Main MDCT program
Idct	Inverse DCT
Imdct	Inverse MDCT

The implementation of init_dct is shown in figure 9.

```
void init_dct(void)
{
   int i,j;
   for(i=0;i<FRAME_SIZE;i++)
   {
      for(j=0;j<FRAME_SIZE;j++)
      {
         dct_table[i][j]=cos( pi * (double)i*(double)j /
(FRAME_SIZE-1) );
      }
   }
}
   }
```

Figure 9. Implementation of Function Init_dct

The implementation of function dct is shown in figure 10.

```
freq_frame *dct(pcm_frame *pcm,freq_frame *freq)
{
   int i,j;
   double acc;

   for(i=0;i<FRAME_SIZE;i++)
   {
      acc=0.5*(double)pcm->buf[0]*dct_table[i][0];
      for(j=1;j<FRAME_SIZE-1;j++)
      {
         acc+=( (double)pcm->buf[j]*
            dct_table[i][j] );
      }
      acc+=0.5*(double)pcm->buf[FRAME_SIZE-
1]*dct_table[i][FRAME_SIZE-1];
      freq->buf[i]=(acc*2.0/(FRAME_SIZE-1));
   }
```

Figure 10. Implementation of Function Discrete Cosine Transform

The implementation of function idct is shown in figure 11.

```
pcm_frame *idct(freq_frame *freq,pcm_frame *pcm)
{
   int i,j;
   double acc;

   for(i=0;i<FRAME_SIZE;i++)
   {
      acc=0.5*freq->buf[0]*dct_table[i][0];
      for(j=1;j<FRAME_SIZE-1;j++)
      {
         acc+=(freq->buf[j]*
            dct_table[i][j] );
      }
      acc+=0.5*freq->buf[FRAME_SIZE-1]*dct_table[i][FRAME_SIZE-1];
      pcm->buf[i]=(short)(acc);
   }
```

Figure 11. Implementation of Function Inverse Discrete Cosine Transform

The implementation of int_mdct is shown in figure 12.

```
void init_mdct(void)
{
   int i,j;
   for(i=0;i<FRAME_SIZE;i++)
   {
      for(j=0;j<FRAME_SIZE*2;j++)
      {
         mdct_ftable[i][j]=2.0*1.0/(FRAME_SIZE-
1)*sin(0.5*pi*(double)j/FRAME_SIZE) *
            cos( ((double)j+(FRAME_SIZE+1.0)/2.0) * ((double)i+0.5
) * pi / FRAME_SIZE);
      }
   }

   for(i=0;i<FRAME_SIZE*2;i++)
   {
      for(j=0;j<FRAME_SIZE;j++)
      {
         mdct_itable[i][j]=1.0*sin(0.5*pi*(double)i/FRAME_SIZE) *
            cos( ((double)i+(FRAME_SIZE+1.0)/2.0) * ((double)j+0.5
) * pi / FRAME_SIZE);
      }
   }
}
```

Figure 12. Implementation of Function Init_mdct

The implementation of mdct is shown in figure 13.

```
freq_frame *mdct(pcm_frame *pcm1,pcm_frame *pcm2,freq_frame *freq)
{
   int i,j;
   double acc;


   for(i=0;i<FRAME_SIZE;i++)
   {
      acc=0.0;
      for(j=0;j<FRAME_SIZE;j++)
      {
         acc+=( (double)pcm1->buf[j]* mdct_ftable[i][j] );
      }
      for(j=0;j<FRAME_SIZE;j++)
      {
         acc+=( (double)pcm2->buf[j]* mdct_ftable[i][j+FRAME_SIZE]
);
      }
      freq->buf[i]=acc;
   }

   return freq;
}
```

Figure 13. Implementation of Function Modified Discrete Cosine Transform

The implementation of imdct is shown in figure 14.

```
pcm_frame *imdct(freq_frame *freq,pcm_frame *pcm1,pcm_frame *pcm2)
{
   int i,j;
   double acc;

   for(i=0;i<FRAME_SIZE;i++)
   {
      acc=0.0;
      for(j=0;j<FRAME_SIZE;j++)
      {
         acc+=(freq->buf[j]* mdct_itable[i][j] );
      }
      pcm1->buf[i]+=(short)(acc);
   }

   for(i=0;i<FRAME_SIZE;i++)
   {
      acc=0.0;
      for(j=0;j<FRAME_SIZE;j++)
      {
         acc+=(freq->buf[j]* mdct_itable[i+FRAME_SIZE][j] );
      }
      pcm2->buf[i]=(short)(acc);
   }

   return pcm2;
}
```

Figure 14. Implementation of Function Inverse Modified Discrete Cosine Transform

4.2.2 Implementation of Genetic Algorithms Training Processes

File "ga.c" contains code to implement the training processes. Three important functions are summarized in table 4.

Table 4. Functions Required for Genetic Training Processes

Function Name	Purpose
mutation	Input chromosome might mutate in the process. If the chromosome mutates, some genes would change their value to zero.
crossover	Generating a new chromosome (child) by combining the genes of two candidates (parents).
evaluate	Calculating the snr value of a chromosome.

The implementation of function mutation is shown in figure 15.

```
void mutation(ga_indiv *pop)
{
   int dis[FRAME_SIZE];
   int i,j,k;

   j=0;
   for(i=0;i<FRAME_SIZE;i++)
   {
      if(pop->sel[i]==0)
         dis[j++]=i;
   }

   for(i=0;i<DISCARD_BANDS;i++)
   {
      if(my_random()<MUT1)
      {
         do {
            k=(my_random()*FRAME_SIZE);
         } while(pop->sel[k]==0);
         pop->sel[k]=0;
         pop->sel[dis[i]]=1;
      }
   }
}
```

Figure 15. Implementation of Function Mutation

The implementation of function crossover is shown in figure 16 and 17.

```
ga_indiv *crossover(ga_indiv *p1,ga_indiv *p2)
{
   int i,j,k,cnt1,cnt2,cnt3;
   int dis1[FRAME_SIZE],dis2[FRAME_SIZE];
   int candidate[FRAME_SIZE],final[FRAME_SIZE];
      ga_indiv *ptr;

      ptr=indiv_create();

   cnt1=0;
   cnt2=0;
   for(i=0;i<FRAME_SIZE;i++)
   {
      if(p1->sel[i]==0)
         dis1[cnt1++]=i;
      if(p2->sel[i]==0)
         dis2[cnt2++]=i;
   }

   cnt3=0;
   for(i=0;i<cnt1;i++)
   {
      int fg1,fg2;

      fg1=1;
      fg2=1;
      for(j=0;j<cnt3;j++)
      {
         if(dis1[i]==candidate[j])
            fg1=0;
```

Figure 16. Implementation of Function Crossover

```
      if(dis2[i]==candidate[j])
            fg2=0;
      }

      if(fg1)
         candidate[cnt3++]=dis1[i];
      if(fg2 && dis2[i]!=dis1[i])
         candidate[cnt3++]=dis2[i];
   }

   for(i=0;i<DISCARD_BANDS;i++)
   {
      j=(my_random()*(double)cnt3);
      final[i]=candidate[j];
      if(j<cnt3-1)
      {
         candidate[j]=candidate[cnt3-1];
      }
      cnt3--;
   }

   printf( "%d\n" , DISCARD_BANDS);
   for(i=0;i<FRAME_SIZE;i++)
      ptr->sel[i]=1;

   for(i=0;i<DISCARD_BANDS;i++)
      ptr->sel[final[i]]=0;

   return ptr;
}
```

Figure 17. Implementation of Function Crossover (Continue)

The implementation of function evaluate is shown in figure 18.

```
double evaluate(ga_indiv *indiv)
{
   int i,j,k;
   double sig,noise,d1,d2;

   sig=0.0;
   noise=0.0;
   for(i=0;i<gaopt->frames;i++)
      for(j=0;j<FRAME_SIZE;j++)
      {
         d1=gaopt->training_data[i]->buf[j];
         if(indiv->sel[j]!=0)
            d2=gaopt->training_data[i]->buf[j];
         else
            d2=0.0;
         sig+=d1*d1;
         noise+=(d1-d2)*(d1-d2);
      }

   indiv->snr=10.0*log10(sig/noise);

   return indiv->snr;
}
```

Figure 18. Implementation of Function Evaluate

4.2.3 Implementation of Psychoacoustic Principles

Psychoacoustic Principles are implemented in function named "freqmask_append" in file freqmask.c. The implementation is shown in figure 19 and 20.

```
void freqmask_append(freqmask *fq,double *data,int *out,double
sample_freq,double k)
{
   double temp, dtemp;
   double bwf, bw;
   int a, b, c, d, e;
   int l_left, r_left;

   fq->spfq=sample_freq;
   bwf=(fq->spfq/2)/fq->fram_size;
   bw=13*atan((0.00076*bwf))+3.5*atan(pow((bwf/7500),2));
   for(a=0;a<fq->fram_size;a++)
   {
      fq->data[a]=fabs(*(data+a))*k;
      temp=(3.64*pow((0.001*(bwf*a)),-0.8)-6.5*exp(-
0.6*((0.001*(bwf*a)-3.3)*(0.001*(bwf*a)-
3.3)))+0.001*pow(0.001*(bwf*a),4));
      fq->qtspl[a]=pow(10,(0.1*temp));
   dtemp=-35.0*freqmask_bark((double)a*bwf);
      fq->sf1[a]=pow(10,(0.1*dtemp));
   dtemp=-35.0*freqmask_bark((double)a*bwf);
      fq->sf2[a]=pow(10,(0.1*dtemp));

      if(fq->data[a]>fq->qtspl[a])
            fq->mdf[a]=1;
      else
            fq->mdf[a]=-1;
   }
for(b=0;b<fq->fram_size;b++)
   {
      l_left=b;
```

Figure 19. Implementation of Function Freqmask_append

```
        r_left=fq->fram_size-b-1;

        c=0;
        while(c<l_left)
        {
                temp=fq->sf1[c+1]*fq->data[b];
                if(temp<=fq->qtspl[b-c])
                        break;
                else
                {
                        if(temp>=fq->data[b-c])
                                fq->mdf[b-c]=0;
                }
                c++;
        }

        for(d=b+1;d<(fq->fram_size);d++)
        {
                temp=fq->sf2[d-b]*fq->data[b];
                if(temp<=fq->qtspl[d])
                        break;
                else
                {
                        if(temp>=fq->data[d])
                                fq->mdf[d]=0;
                }
        }
    }
for(e=0;e<fq->fram_size;e++)
        out[e]=fq->mdf[e];

}
```

Figure 20. Implementation of Function Freqmask_append (Continue)

CHAPTER FIVE

PERFORMANCE ANALYSIS

This chapter focuses on the performance of techniques used in the project. Two major filter techniques used in filter analysis are discussed in 5.1. The differences and performance of psychoacoustic principles and genetic algorithm will be illustrated in 5.2. Section 5.2 also introduces two audio compression systems, MP3 and Vorbis, and compares their performance with the result of the project.

5.1 Filter Analysis

Human ear is not sensitive enough to detect the differences of all the audio signal information, especially when the difference is small. In order to produce a better compression product, a filter is developed to detect the difference between similar signals. By using a filter, sound signal is transformed from time-domain to frequency-domain. Discrete cosine transform (DCT) and modified discrete cosine transform (MDCT), two Fourier-related transforms, are the most common techniques used in audio compression to help compress sound signals.

5.1.1 Discrete Cosine Transform and Modified Discrete Cosine Transform

There are many different types of DCT. They range from DCT-1 to DCT-4. DCT-1 is used in this project. The formula of DCT-1 is defined by the formula

$$X[k] = 2\sum_{n=0}^{N-1} \alpha[n]x[n]\cos\left(\frac{\pi kn}{N-1}\right) \qquad 0 \le k \le N-1$$

the inverse of DCT-1 is defined by the following formula

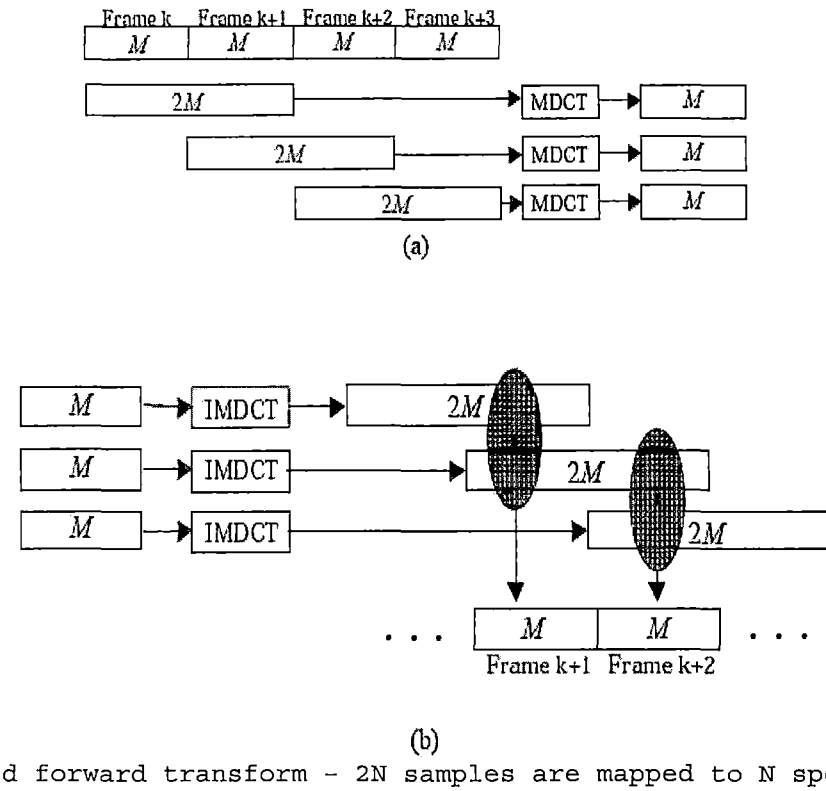
$$x[n] = \frac{1}{N-1}\sum_{k=0}^{N-1} a[k]X[k]\left(\cos\frac{\pi kn}{N-1}\right) \qquad 0 \le n \le N-1$$

when n=0 or N-1, $\alpha[n]$=0.5; otherwise, $\alpha[n]$=1.

DCT is used in JPEG audio compression, MJPEG video compression and MPEG video compression. In Mathematics, DCT is also used to solve partial differential equation.

Modified discrete cosine transform, or MDCT, is a variant of DCT-4. MDCT is used in AAC, MP3 and Vobis audio compression. By applying lapped transform. MDCT has half as many outputs as inputs. The inverse MDCT is known as the IMDCT. In general, MDCT is not invertible. However, perfect invertibility is achieved by adding the overlapped blocks, causing the errors to cancel and the original data to be retrieved; this technique is known as time-domain aliasing cancellation (TDAC). The IMDCT transforms n real numbers into 2n real numbers. The overlapped analysis and

overlap-add synthesis processes are illustrated in figure 21.



(a) Lapped forward transform - 2N samples are mapped to N spectral components. (b) Inverse transform - N spectral components are mapped to vector of 2N samples

Figure 21. Modified Discrete Cosine Transform

The definition of MDCT is

$$X(m) = \sum_{k=0}^{n-1} f[k]x[k]\cos\left(\frac{\pi}{2n}\left(2k+1+\frac{n}{2}\right)(2m+1)\right) \quad \text{, for m= } 0\ldots\frac{n}{2}-1$$

and the IMDCT:

$$y(p) = f(p)\frac{4}{n}\sum_{m=0}^{\frac{n}{2}-1} X(m)\cos\left(\frac{\pi}{2n}\left(2p+1+\frac{n}{2}\right)(2m+1)\right) \quad \text{, for p= } 0\ldots n-1$$

where f(x) is a function with certain properties. The sine equation

$$f(x) = \sin\left(\pi \frac{x}{n}\right)$$

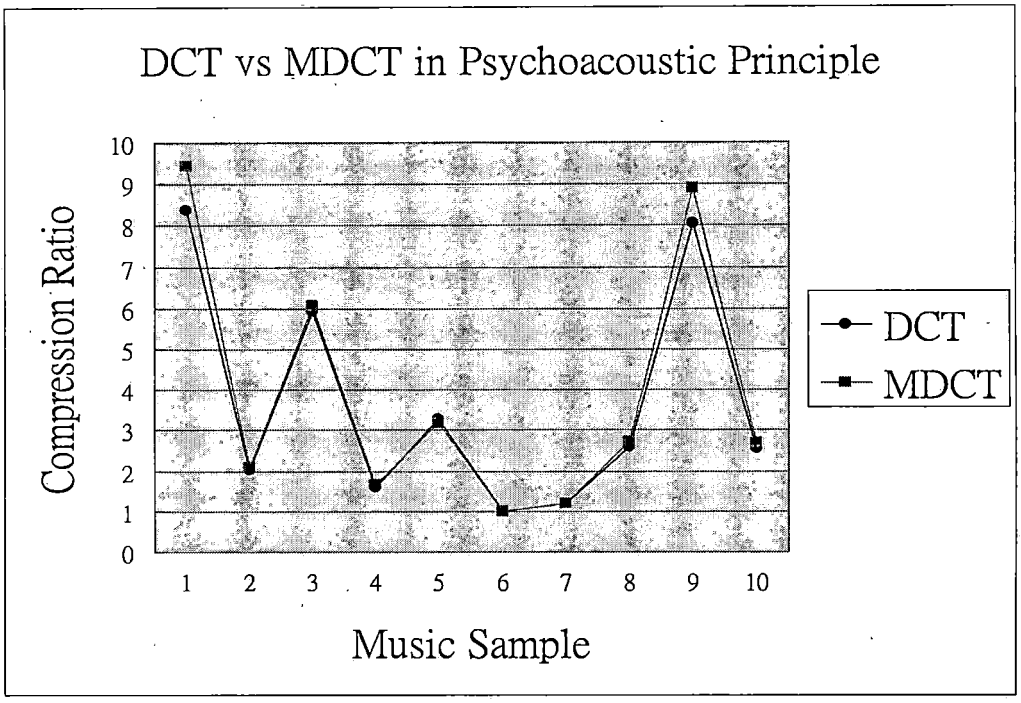
has the right properties.

5.1.2 Complexity of Discrete Cosine Transform and Modified Discrete Cosine Transform

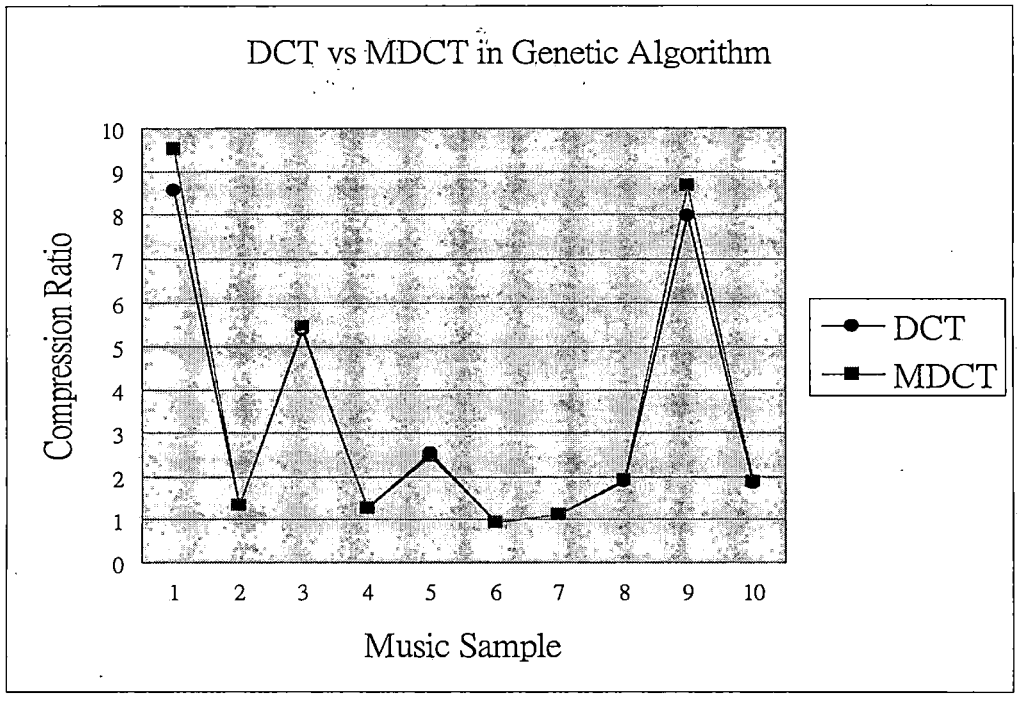
The direct application of the DCT or MDCT requires O(n2) operations. However, by factorizing the computation, one can compute the same thing with only $O(n^{\log}n)$. MDCT can be computed in $O(n^{\log}n)$ by recursively factorizing the computation. The difference between O(n2) and $O(n^{\log}n)$ is significant when the data size becomes large.

5.1.3 Performance of Discrete Cosine Transform and Modified Discrete Cosine Transform

Ten music samples were selected as sample to check the performance of DCT and MDCT. The compression ratios of these two methods are graphed in Figure 22. Figure 22(a) employs psychoacoustic principles while Figure 22(b) applies genetic algorithms.



(a)



(b)

DCT and MDCT are used to compress ten music file.
(a) Psychoacoustic principles (b)Genetic algorithms

Figure 22. Discrete Cosine Transform versus Modified Discrete Cosine Transform

Figure 22 shows the performance of DCT and MDCT is about the same. MDCT is slightly better when the compression ratio is high, but, for cases of low compression ratio, they perform equally well.

5.2 Comparing Psychoacoustic Principles with Genetic Algorithm

This section will discuss the performance of psychoacoustic principles and genetic algorithm. Signal-to-noise ratio and subjective testing will be used as the measurements.

5.2.1 Signal-to-Noise Ratio

Signal-to-noise ratio [13], or SNR, is a measure of signal strength relative to background noise. The measure is usually measured in decibels (dB). Let S1i be the signal of original music file and S2i be the signal of decompressed file. SNR is defined in the following formula.

$$\mathrm{SNR} = 20\log_{10}\left(\frac{\sum_{i=1}^{N}(S1i)^2}{\sum_{i=1}^{N}(S1i - S2i)^2} \right)$$

Compression Ratio	Original File Size	SNR for Psychoacoustic Principles	SNR for Genetic Algorithms
2.7	198KB	7.61	9.18
4.27	5.04MB	15.67	6.91
9.44	5.04MB	19.27	18.62
6.06	5MB	18.44	11.62
8.89	73.6KB	19.06	20.52

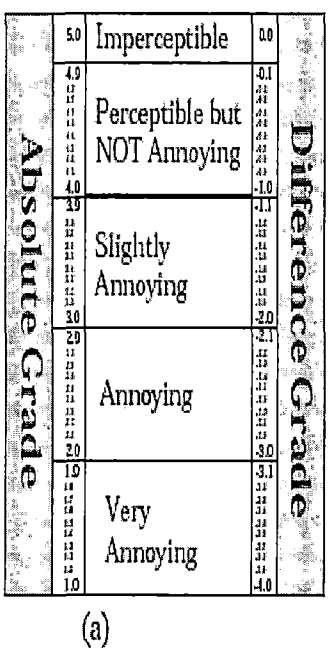
Figure 23. Comparing Psychoacoustic Principle with Genetic Algorithm Using Signal-to-Noise Ratio

From the testing result in Figure 23, we can conclude that psychoacoustic principle performs better than genetic algorithms in most cases. However, SNR is totally based on mathematics and statistic result. It is noticed that in some cases, human ear can easily detect the noise of a decompressed file with high SNR. The listeners may not detect noise in the music files with low SNR. Because of this, subjective quality measures are designed to give a more reliable result.

5.2.2 Subjective Quality Measure

Subjective quality measures can be roughly divided into two categories, absolute category rating (ACR) and

comparison category rating (CCR). Figure 24 shows two subjective quality scales.



CCR	
A better than B	+2
A slightly better than B	+1
A same as B	0
A slightly worse than B	-1
A worse than B	-2
A much worse than B	-3

(b)

(a) Absolute Category Rating (ACR)
(b) Comparison Category Rating (CCR)

Figure 24. Subjective Quality Scales

During the absolute category rating process, uncoded signal is played first as reference. Encoded signal and uncoded signal would be played in a random order. After listening all, the subject must identify which one (second one or last one) is the hidden reference. Subjects give the impaired stimulus (coded signal) relative to the reference using the 41-point, 5 grade score. Grade 5 is reserved for the signal that subject believes identical with the reference.

Comparison category rating (CCR) is relatively simple. Investigators prefer CCR over 5-point ACR.

Five of the testing results for subjective testing are selected and presented in Figure 25. The score given by each listener for the test is listed in Appendix B.

Compression Ratio	Original File Size	Codec 1	SNR for Codec1	Codec 2	SNR for Codec2	Average Subjective Testing
2.70	198KB	Psychoacoustic principle	7.61	Genetic Algorithms	9.18	1.33
4.27	5.04MB	Genetic Algorithms	6.91	Psychoacoustic principle	15.67	-2.17
9.44	5.04MB	Psychoacoustic principle	19.27	Genetic Algorithms	18.62	0.06
6.06	5MB	Genetic Algorithms	11.62	Psychoacoustic principle	18.44	-0.11
8.89	73.6KB	Genetic Algorithms	20.52	Psychoacoustic principle	19.06	-0.06

Figure 25. Subjective Testing Result of Psychoacoustic Principle and Genetic Algorithms

From the result, we can conclude psychoacoustic principle is better than genetic algorithms. One reason that causes this result is genetic algorithm does not analyze every input file. Instead, it only compresses the source file based on the subbands selected during the training processes. Genetic algorithms run slightly faster, but the performance is not as good as psychoacoustic principle.

5.3 Comparing Azip with Moving Picture Experts Group-Layer3 and Ogg Vorbis

5.3.1 Background

MP3 was developed around 1993 and became very popular in 1995. In the first half of 1995, there were a large number of MP3 files on the Internet. MP3 is an audio compression algorithm capable of greatly reducing the amount of data required for audio file. Today, many new and advanced algorithms, such as advanced audio coding (AAC), LAME, and Ogg Vorbis, have been developed, but MP3 is still a common audio compression format.

In September 1998, Fraunhofer Gesellschaft, the inventor of MP3, planned to charge license fee for the MP3 format and eventually he did. As a result, Christopher Montgomery began to work on the Ogg project. A stable version of the codec was released on July 2002.

Ogg Vorbis was a free and open audio compression codec. It uses the MDCT to convert signals from time domain to the frequency domain. Vector quantization was applied during the quantization stage to achieve the best result.

5.3.2 Comparing with Ogg and Moving Picture Experts Group-Layer3

MP3 and Ogg are compared in many websites and research reports [4]. Most research papers conclude that

Ogg records better than MP3. It is noticed that MP3 model deviates from the original data more strongly than OGG at higher frequencies. According to Aziz H. Poonawalla [4], Ogg performs better than MP3. Figure 26 shows the SNR and subjective testing of 5 selected pairs[1]. The testing result shows Ogg performs the best. This also indicates there are still some areas where Azip can be improved. Improvement that can be made is discussed in 6.1.

Compression Ratio	Original File Size	Codec 1	SNR for Codec1	Codec 2	SNR for Codec2	Average Subjective Testing
8.65	198KB	Psychoacoustic Principle	7.23	MP3	11.02	-0.06
11.8	5.04MB	Ogg	10.77	Psychoacoustic Principle	14.09	0.11
13.79	14.6MB	Ogg	22.94	Genetic Algorithms	10.2	0.06
44	11MB	Genetic Algorithms	3.42	MP3	25.86	-1.72
13.99	6.47MB	Ogg	21.22	Psychoacoustic Principle	18.56	0.06

Figure 26. Subjective Testing Result of Moving Picture Experts Group-Layer3 and Ogg

[1] The score given by each listener for each test is listed in Appendix C

CHAPTER SIX

FUTURE ENHANCEMENT AND CONCLUSION

6.1 Vector Quantization

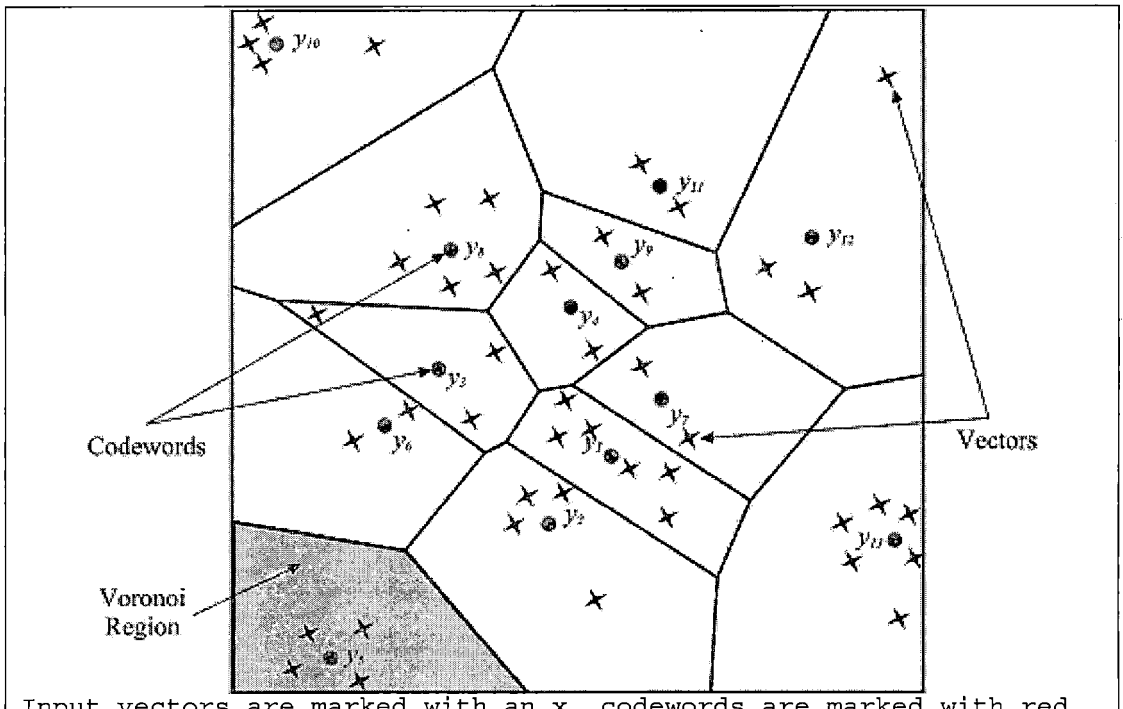
Vector quantization is an algorithm used in voice recognition, audio and video compression [3]. Many advanced audio compression applications, such as Vorbis Ogg, use vector quantization instead of scalar quantization.

In scalar quantization, one represents the value by fixed subset of representative values. For instance, a 16-bit value may only be represented by eight most significant bits. As the result, we only get an approximation of the original data.

Although vector quantization is also a lossy compression, it significantly reduces the difference between the original and the output number. Vector quantization breaks all numbers into several regions. For each region, a representative codeword is chosen to represent all numbers in the region. The representative codeword is determined to be the closest in Euclidean distance from the input vector. The Euclidean distance is defined by:

$$d(x, y_i) = \sqrt{\sum_{j=1}^{k} (x_j - y_{ij})^2}$$

where x_j is the jth component of the input vector, and y_{ij} is the jth is component of the codeword y_i. Figure 27 illustrates how vector quantization works in 2-dimensional space.



Input vectors are marked with an x, codewords are marked with red circles, and the Voronoi regions are separated with boundary lines.

Figure 27. Codewords in 2-Dimensional Space

6.2 Conclusion

Azip audio compression system is a text-based application. It is capable of compressing .wav files, decompressing .csusb files and comparing the SNR of two

music files. Moreover, by changing the configuration, users can compare the difference between DCT and MDCT. It also provides an interface for users who intend to study the application of psychoacoustics principle and genetic algorithms in audio compression. The theory of psychoacoustics principle and genetic algorithms were discussed in chapter 2 and chapter 3.

The performance of DCT and MDCT was discussed in chapter 5. The test result shows that MDCT performs slightly better especially when the input file is large. (over 1 MB) Otherwise, DCT performs as well as MDCT.

Psychoacoustic principle is widely applied in all audio compression applications. Psychoacoustic principle was compared with genetic algorithm in chapter 5. The result shows that psychoacoustic principle does work better than genetic algorithm. Azip audio compression system is compared with MP3 and Vorbis Ogg. Vorbis Ogg works much better than Azip audio compression system. This shows that there are still some areas that can be improved.

During the implementation of this project, a number of ideas became clearer. Starting from the format of .wav file, the algorithms using for audio compression, the implementation of Azip, to the comparison between Azip

with other audio compression system sold in the market or downloaded from internet, I absorbed a lot of knowledge from reading the articles related to audio compression. In each stage, I have a better understanding in audio compression. Unlike lossless compression used in text-form data compression, good algorithms used in audio compression require the understanding of physics and the structure of human ear. Once again, I would like to thank all people who help me understand the algorithms and answer the questions I have during the project.

APPENDIX A

ABBREVIATION

The appendix lists all abbreviation used in the document.

AAC	Advanced Audio Coding
ACR	Absolute Category Rating
CCR	Comparison Category Rating
CSUSB	California State University San Bernardino
DCT	Discrete Cosine Transform
GA	Genetic Algorithms
IDCT	Inverse Discrete Cosine Transform
IMDCT	Inverse Modified Discrete Cosine Transform
JPEG	Joint Photographic Experts Group
MDCT	Modified Discrete Cosine Transform
MP3	MPEG Layer3
MPEG	Moving Picture Experts Group
NMN	Noise-Masking-Noise
NMT	Noise-Masking-Tone
SMR	Signal-to-Mask Ratio
SNR	Signal-to-Noise Ratio
SPL	Sound Pressure Level
TDAC	Time-Domain Aliasing Cancellation
TMN	Tone-Masking-Noise

APPENDIX B

COMPARING PSYCHOACOUSTIC PRINCIPLES WITH GENETIC ALGORITHMS

The appendix lists the scores given by listeners for the subjective testing for comparing psychoacoustic principles with genetic algorithms.

Listener	Test 1	Test 2	Test 3	Test 4	Test 5
1	1	-3	0	0	0
2	2	-2	0	-1	1
3	1	-2	0	0	0
4	1	-2	1	0	1
5	1	-2	-1	0	1
6	2	-3	1	1	0
7	1	-2	0	0	-1
8	1	-2	0	-1	0
9	1	-2	0	0	-1
10	1	-3	-1	0	0
11	1	-2	0	-1	1
12	2	-2	1	1	-1
13	1	-2	0	0	0
14	2	-2	-1	1	-1
15	1	-1	0	0	0
16	1	-2	0	-1	0
17	2	-3	1	-1	-1
18	2	-2	0	0	0
Average	1.333333	-2.166667	0.055556	-0.111111	-0.055556

The grade is given using Comparison Category Rating (CCR). Grade given can be ranged from 2(the file played first is much better) to -3(the file played first is much worse). 0 indicates the listener feel two files are the same.

APPENDIX C

COMPARING AZIP WITH MOVING PICTURE EXPERTS GROUP-LAYER3 AND OGG

The appendix lists the scores given by listeners for the subjective testing for comparing Azip with MP3 and Ogg.

Listener	Test 1	Test 2	Test 3	Test 4	Test 5
1	-1	0	0	-1	-1
2	1	1	1	-2	1
3	0	0	0	-1	0
4	0	0	0	-1	0
5	-1	0	0	-2	0
6	0	0	0	-1	0
7	1	1	1	-3	1
8	-1	0	-1	-1	1
9	0	0	0	-1	0
10	0	0	0	-2	0
11	1	0	0	-3	0
12	0	-1	2	-1	1
13	0	0	0	-2	0
14	0	-1	-1	-1	-1
15	0	0	0	-2	0
16	-1	0	-1	-2	-1
17	0	1	0	-3	0
18	0	1	0	-2	0
Average	-0.055556	0.1111111	0.0555556	-1.722222	0.0555556

The grade is given using Comparison Category Rating (CCR). Grade given can be ranged from 2(the file played first is much better) to -3(the file played first is much worse). 0 indicates the listener feel two files are the same.

REFERENCES

[1]. F. Wylie, "Digital Audio Data Compression", Electronics & Communication Engineering Journal, February 1995

[2]. Remy Boyer and Karim Abed-Meraim, Member, IEEE, "Audio Modeling Based on Delayed Sinusoids", IEEE Transactions on Speech and Audio Processing, Vol 12, No.2, March 2004

[3]. "Vector Quantization" http://www.geocities.com/mohamdqasem/vectorquantization/vq.html

[4]. Aziz H. Poonawalla "MP3 and OGG Audio Compression", August 2001 http://www.abde.net/projects/ogg_MP3

[5]. Ted Painter and Andreas Spanias, "Perceptual Coding of Digital Audio", Poceddings of the IEEE. VOL 88. No. 4 April 2000 http://www.eas.asu.edu/~spanias/papers/paper-audio-tedspanias-00.pdf

[6]. P.Menardi, G.A.Mian, G.Riccardi, "Dynamic Bit Allocation in Subband Coding of Wideband Audio with Multipulse LPC"

[7]. Rongshan Yu and C. C. Ko,"A Warped Linear-Prediction-Based Subband Audio Coding Algorithm", IEEE Transactions on Speech and Audio Processing, VOL. 10, NO.1, January 2002 http://www.ee.columbia.edu/~marios/courses/e6820y02/project/papers/A%20Warped%20Linear-Prediction-Based%20Subband%20Audio%20Cording%20Algorithm.pdf

[8]. "Image Compressing - from DCT to Wavelets : A Review" Subhasis Saha http://www.acm.org/crossroads/xrds6-3/sahaimgcoding.html

[9]. "A&E's Technical Guides to All Things Audio and Video" http://www.animemusicvideos.org/guides/avtech

[10]. Ken C. Pohlmann, "Digital Audio CD and Other Selected Digital Technologies Based on Principles of Digital Audio, 4th Ed." http://www.mcgoodwin.net/digitalaudio/digitalaudio.html

[11]. Seymour Shlien and Gilbert Soulodre, "Measuring the Characteristeics of "Expert" Listeners" http://www.MP3-tech.org/programmer/docs/96-05.pdf

[12]. "Audio Compression" http://www.cs.sfu.ca/CourseCentral/365/li/material/notes/Chap4/Chap4.4/Chap4.4.html

[13]. "signal-to-noise ratio" http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci213018,00.html

[14]. G. Syswerda. "Uniform Crossover in Genetic Algorithms", In J. D. Schaffer, editor, Proc. 3rd Int'l Conf. on Genetic Algorithms, pages 2-9, San Mateo, CA, 1989. Morgan Kaufmann.

[15]. Alan V. Oppenheim and Ronald W. Schafer, "Discrete-Time Signal Processing" Second Edition, Prentice-Hall, Inc. pages 591.Upper Saddle River, NJ, 1989.

[16]. Jenq-Shyang Pan, "Improved Algorithms For VQ Codeword Search, Codebook Design and Codebook Index Assignment", Doctoral Dissertation, Science and Engineering of Edinburgh, 1996.