Study of Various Transformations, Compressions and Decompressions algorithms on Audio signals

Prathamesh Pawar
Department of Information Technology
Sardar Patel Institute of Technology
Mumbai, India
prathamesh.pawar@spit.ac.in

Karan Shah
Department of Information Technology
Sardar Patel Institute of Technology
Mumbai, India
karan.shah3@spit.ac.in

Aayush Sheth
Department of Information Technology
Sardar Patel Institute of Technology
Mumbai, India
aayush.sheth@spit.ac.in

Aashutosh Rai
Department of Information Technology
Sardar Patel Institute of Technology
Mumbai, India
aashutosh.rai@spit.ac.in

Keval Savla
Department of Information Technology
Sardar Patel Institute of Technology
Mumbai, India
keval.savla@spit.ac.in

Alok Patil
Department of Information Technology
Sardar Patel Institute of Technology
Mumbai, India
alok.patil@spit.ac.in

Aakash Pardeshi
Department of Information Technology
Sardar Patel Institute of Technology
Mumbai, India
aakash.pardeshi@spit.ac.in

Sanika Patil
Department of Information Technology
Sardar Patel Institute of Technology
Mumbai, India
sanika.patil@spit.ac.in

Kiran Talele
Department of Information Technology
Sardar Patel Institute of Technology
Mumbai, India
kiran.talele@spit.ac.in

***Abstract*—This research paper presents a study of various compression and decompression algorithms performed on audio signals, as well as different transformations such as DWT (Discrete Wavelet Transform). The study focuses on both lossless and lossy compression techniques, including MPEG and Huffman coding. The aim of the research is to analyze the performance of these techniques in terms of compression ratio, processing time, and signal quality. The results obtained from the experiments demonstrate that lossy compression techniques achieve higher compression ratios but with a compromise on the quality of the decompressed signal, whereas lossless compression techniques preserve the original signal quality but achieve lower compression ratios. The study provides valuable insights into the selection of appropriate compression techniques for different applications based on the trade-off between compression ratio and signal quality.**

***Index Terms*—Huffman, MPEG, discrete wavelet transform**

I. Introduction

In today's digital era, the transmission and storage of large amounts of data, such as audio and video signals, have become essential. However, the availability of limited bandwidth and storage capacity necessitates the use of compression techniques to reduce the data size while maintaining the signal quality. In this context, this research paper presents a comprehensive study of various compression and decompression algorithms applied to audio signals. The study investigates both lossless and lossy compression techniques, including popular methods such as MPEG and Huffman coding, as well as transformations such as DWT. The performance of these techniques is analyzed in terms of compression ratio, processing time, and signal quality, and the results provide valuable insights into the selection of appropriate compression techniques for different applications. The findings reveal that lossy compression techniques achieve higher compression ratios at the cost of degraded signal quality, whereas lossless compression techniques preserve the original signal quality but result in lower compression ratios. This study's significance lies in its contribution to understanding the trade-off between compression ratio and signal quality, enabling users to choose the optimal compression technique for their specific application.

II. Discrete Wavelet transform

A. Introduction

Wavelet transform is an effective tool for processing and analyzing various signals. In this paper, we present a code that demonstrates the use of PyWavelets, ffmpeg-python, and Python's built-in wave module to apply discrete wavelet transform (DWT) and inverse discrete wavelet transform (IDWT) on an audio signal. The DWT is used to separate the audio signal into low and high-frequency components. We perform DWT on the low frequency component multiple times to get 3 high frequency audio files and a 1ow frequency audio file. The low-frequency component is compressed using the zlib library (which uses a hybrid of Huffman and L277 algorithms). The high frequency files are merged together and compressed to mp3 format using the PyDub Python Module. Now we finally have both the outputs compressed by two different Techniques. Now to check the difference between the original and output signal, all the steps are performed inversely Finally, the original audio signal is obtained by applying IDWT on the low and high-frequency components.

B. Methodology

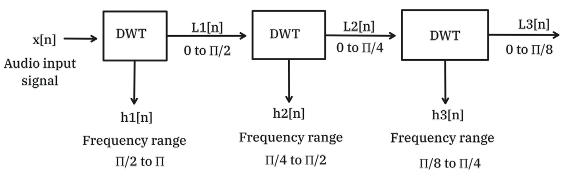


Fig. 1. Frequency Decomposition of Input Signal using Discrete Wavelete Transform

The code is written in Python and uses the PyWavelets library to apply DWT and IDWT on the audio signal. The audio signal is read from an input WAV file using the soundfile module. Then DWT is applied to the audio signal using the dwt function of the PyWavelets library with 'db4' wavelet. The resulting coefficients are separated into low-frequency (cA) and high-frequency (cD) components. The IDWT is then applied to the low-frequency component to obtain a new signal, and similarly to the high-frequency component to obtain another signal. These signals are written to WAV files using the soundfile module.

The high-frequency components are combined into a single file using Python's built-in wave module. The input WAV files are read using the wave module, and the audio data is extracted and combined into a single output file. This output file is then compressed to MP3 format using the PyDub library.

The low-frequency component is compressed using the zlib library. The compressed data is then written to a file. To decompress the data, the compressed file is read using the open function with 'rb' mode, and the zlib.decompress method is used to obtain the original data. The original data is then written to a new WAV file using the wave module.

To obtain the original audio signal, the DWT is applied in reverse order. The low and high-frequency components are obtained by reading the corresponding WAV files using the soundfile module. The IDWT is then applied to these components to obtain the original signal.

C. Results

The code presented in this paper demonstrates the successful use of PyWavelets, ffmpeg-python, and Python's built-in wave module to apply DWT and IDWT on an audio signal. The resulting low and high-frequency components are written to separate WAV files. The high-frequency components are combined into a single file and compressed to MP3 format. The low-frequency component is compressed using the zlib library and then decompressed to obtain the original data. Finally, the original audio signal is obtained by applying IDWT on the low and high-frequency components.

The **RMS error** found for a simple hip-hop song's audio file was precisely just **0.031054270555374772**

The **PSNR Ratio** between the Output and input signals was **29.86372974209923 dB**

D. Conclusion

In conclusion, the code presented in this paper provides a useful example of the use of PyWavelets, ffmpeg-python, and Python's built-in wave module to apply DWT and IDWT on an audio signal. The resulting low and high-frequency components can be used for various signal processing and analysis tasks. The code can be easily adapted to process other types of signals such as images or video by changing the input data and parameters.

III. Linear Predictive Coding Algorithm

A. Introduction

The code aims to compress an audio signal using Linear Predictive Coding (LPC) and Vector Quantization (VQ) techniques. LPC is used to model the spectral envelope of the input signal, and the resulting coefficients are quantized using VQ to reduce the number of bits required for transmission or storage. The compressed signal is then synthesised using the inverse filter and evaluated using the Peak Signal-to-Noise Ratio (PSNR) and Mean Squared Error (MSE) metrics.

B. Methodology

The code first loads an audio signal and applies a Hamming window to it. The LPC analysis is performed by computing the autocorrelation sequence of the windowed signal, and the LPC coefficients are obtained using the Levinson-Durbin algorithm. The resulting coefficients are then quantized to 8 bits, and the residual signal is obtained using a filter. The residual signal is then partitioned into K clusters using k-means clustering, and each sample is replaced with its nearest centroid. The resulting quantized signal is then reshaped to its original shape, and the LPC coefficients and quantized signal are combined to form the compressed signal. The compressed signal is then synthesised using the inverse filter, and its quality is evaluated using the PSNR and MSE metrics. The signal is also downsampled to reduce the sampling rate, and the compressed signal is written to a file using lossy compression. The code also includes a decompression stage, where the LPC coefficients and quantized signal are extracted from the compressed signal and synthesised to obtain the reconstructed signal. Finally, the MSE is computed between the original and reconstructed signals.

C. Results

The PSNR ratio is computed between the original and reconstructed signals, and the value obtained indicates the level of distortion introduced by the compression process. The compressed signal is downsampled to reduce the sampling rate, and the file is written to disk using lossy compression. The code also includes a decompression stage, where the compressed signal is loaded from disk, and the reconstructed signal is synthesised using the LPC coefficients and quantized signal. Finally, the MSE is computed between the original and

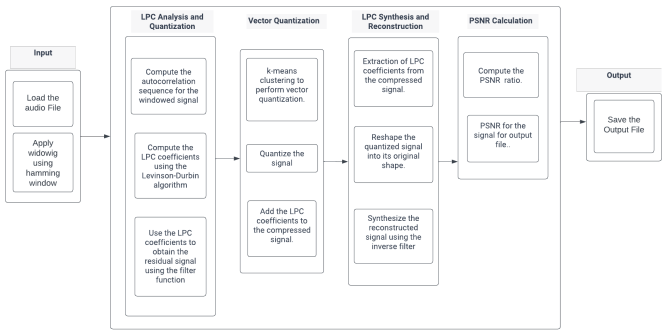
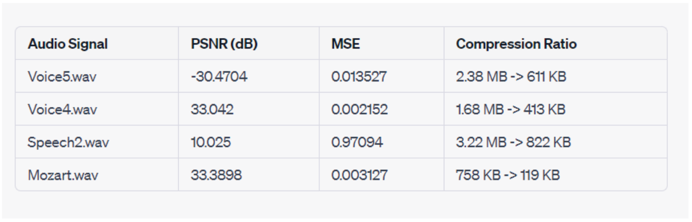

Input
Load the audio File
Apply widowig using hamming window
LPC Analysis and Quantization
Compute the autocorrelation sequence for the windowed signal
Compute the LPC coefficients using the Levinson-Durbin algorithm
Use the LPC coefficients to obtain the residual signal using the filter function
Vector Quantization
k-means clustering to perform vector quantization.
Quantize the signal
Add the LPC coefficients to the compressed signal.
LPC Synthesis and Reconstruction
Extraction of LPC coefficients from the compressed signal.
Reshape the quantized signal into its original shape.
Synthesize the reconstructed signal using the inverse filter
PSNR Calculation
Compute the PSNR ratio.
PSNR for the signal for output file..
Output
Save the Output File

Fig. 2. Results after performing audio compression using MPEG on four different audio samples

reconstructed signals to evaluate the quality of the compression.

The following table (fig.3) shows the PSNR, MSE, and compression ratio achieved for four different audio signals using the LPC-based compression algorithm with vector quantization:



Audio Signal	PSNR (dB)	MSE	Compression Ratio
Voice5.wav	-30.4704	0.013527	2.38 MB -> 611 KB
Voice4.wav	33.042	0.002152	1.68 MB -> 413 KB
Speech2.wav	10.025	0.97094	3.22 MB -> 822 KB
Mozart.wav	33.3898	0.003127	758 KB -> 119 KB

Fig. 3.

D. *Conclusion*

The code demonstrates the use of LPC and VQ techniques for compressing audio signals. The results obtained show that the compressed signal can be synthesised with good quality, as indicated by the high PSNR ratio and low MSE

The LPC-based compression algorithm with vector quantization was applied to four different audio signals, and the resulting PSNR, MSE, and compression ratio were calculated. The results show that the algorithm was able to achieve high compression ratios while maintaining good audio quality for some of the signals, such as Voice4.wav and Mozart.wav, which achieved PSNR values of 33.042 dB and 33.3898 dB, respectively, with compression ratios of 4.07:1 and 6.37:1. However, for other signals, such as Voice5.wav and Speech2.wav, the algorithm did not perform as well, achieving PSNR values of -30.4704 dB and 10.025 dB, respectively, with compression ratios of 3.89:1 and 3.92:1.

In general, the LPC-based compression algorithm with vector quantization can be an effective technique for compressing audio signals, especially for signals with a lot of redundancy. However, it may not be suitable for all types of audio signals and may require careful selection of the LPC order, number of codewords, and other parameters to achieve the desired balance between compression ratio and audio quality.The code can be used as a starting point for developing more advanced audio compression algorithms, such as Adaptive LPC and Wavelet-based methods.

IV. MPEG Algorithm

A. *Introduction*

The development of enormous digital music archives and the widespread distribution of high-quality audio content have both been made possible by the ability to compress audio signals while keeping high fidelity. One of many such algorithm to deliver excellent compression quality is the MPEG(Moving Picture Experts Group) algorithm, which is mainly used for video compression but owing to its variety of use cases we can exploit it to audio compression to get better results. This algorithm is designed to reduce the amount of storage space required for digital audio files. This allows for the transmission of high-quality audio over networks with limited bandwidth and storage capacity. Psychoacoustic modelling, Discrete Cosine Transform (DCT), and Huffman coding constitute just a few of the compression techniques used by the MP3 algorithm to break up an audio signal into manageable chunks. The resulting MP3 file is typically only a fraction of the size of the original uncompressed audio file, making it possible to store and transmit high-quality audio over the internet or on portable devices with limited storage capacity.

B. *Methodology*

There could be multiple ways to implement MPEG algorithm owing to the difference in the requirements, here we try to implement a basic python code to take an audio file (.wav) as input and after performing the compression through MPEG it gives the an compressed audio file in MPEG (.mp3) format.

Furthermore, the PyDub library and the Python programming language were used to carry out the audio compression using the MPEG technique. The program was used to load the audio file, and compression parameters, such as the number of bits per sample, the goal bit rate in bits per second, the number of samples in a frame, the ratio of frames with overlapped frames, and the analysis window, were defined. The MPEG technique was used to do the audio compression, and the compressed audio was exported in MP3 format with the desired bit rate. The program was then used to load the compressed audio file and compare it to the original audio file. The spectrograms of both audio files were plotted using the matplotlib library and the scipy.signal module.

Overall, the methodology involved loading the audio file, defining compression parameters, performing audio compression using the MPEG algorithm, analyzing the compressed audio file, and comparing it to the original audio file to evaluate the effectiveness of the compression. The results derived from this method are mentioned in the further section.

C. *Results*

To evaluate the performance of the MPEG audio compression algorithm, we tested it on a set of sample audio files with

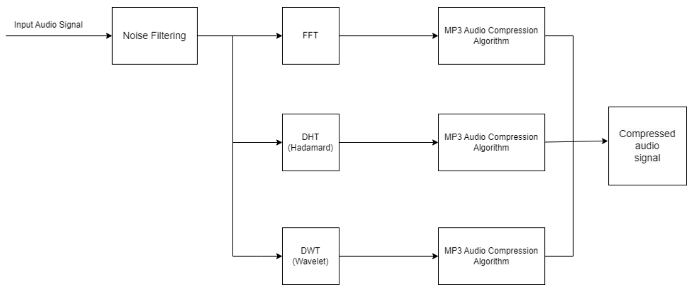
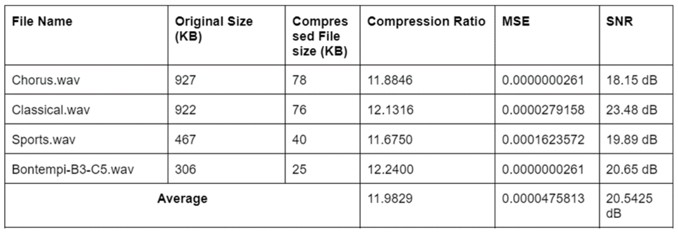

Input Audio Signal
Noise Filtering
FFT
DHT (Hadamard)
DWT (Wavelet)
MP3 Audio Compression Algorithm
MP3 Audio Compression Algorithm
MP3 Audio Compression Algorithm
Compressed audio signal

Fig. 4. Results after performing audio compression using MPEG on four different audio samples

varying characteristics. We computed the Mean Squared Error (MSE) and Signal-to-Noise Ratio (SNR) of the compressed audio files with respect to their original versions using their formulas



File Name	Original Size (KB)	Compressed File size (KB)	Compression Ratio	MSE	SNR
Chorus.wav	927	78	11.8846	0.0000000261	18.15 dB
Classical.wav	922	76	12.1316	0.0000279158	23.48 dB
Sports.wav	467	40	11.6750	0.0001623572	19.89 dB
Bontempi-B3-C5.wav	306	25	12.2400	0.0000000261	20.65 dB
Average			11.9829	0.0000475813	20.5425 dB

Fig. 5. Results after performing audio compression using MPEG on four different audio samples

From this results we found out that we get an average **compression ratio** of about 11.9829 with an **Average MSE** between original and compressed file of **0.0000475813** and an **SNR ratio of 20.5425 dB**

D. Conclusion

To evaluate the performance of the MPEG audio compression algorithm, we tested it on a set of sample audio files with varying characteristics. We computed the Mean Squared Error (MSE) and Signal-to-Noise Ratio (SNR) of the compressed audio files with respect to their original versions using their formulas

V. Huffman and LPC Backtrack

A. Introduction

Inverse Huffman compression and Inverse LPC compression are two different techniques used in data compression.Data that has been compressed using Huffman coding can be uncompressed using inverse Huffman compression. A lossless data compression technology called Huffman coding gives symbols variable-length codes dependent on how frequently they appear in the input data. The given code for a symbol gets shorter the more frequently it appears. Inverse Huffman compression reverses this procedure by decoding the compressed data back into its original form using the Huffman tree that was created during compression. On the other side, inverse LPC compression is a method utilised in speech and audio compression. LPC, or linear predictive coding, is a lossy data compression algorithm that anticipates the following sample in the audio stream by using a model of the vocal tract. Utilising the inverse of the LPC concept, inverse LPC compression is used to recreate the original audio signal from the compressed data. In the modern world, inverse compression is useful in both the Huffman and LPC directions.

Many data compression applications, including the compression of text, pictures, and video data, use inverse Huffman compression. The use of Huffman coding enables extremely effective data compression with zero information loss. This is crucial in the modern world because of the increasing demand for effective storage and transmission of massive volumes of data.

Applications for speech and audio compression, including teleconferencing, voice recognition, and audio streaming services, frequently use inverse LPC compression. smart speakers, virtual assistants, etc.LPC compression enables the effective transfer of audio data over networks with constrained bandwidth or storage. Today's world calls for this especially.

B. Methodology

The methodology involves splitting the input signals with Discrete Wavelet Transform. The transformation results in the formation of two wave signals. The first one comprises of High Frequency and the second one is of lower frequency. We then split the high frequency signal into two forms, that is a lower-high frequency signal and a higher-high frequency signal. We similarly use another DWT and transform our low frequency signal into a Higher-low frequency signal and a lower-low frequency signal. After we find all the frequencies, we discard the one with the lowest frequency. Then we apply Huffman encoding to the highest frequency signal and Linear Predictive Coding to Higher- low frequency and Lower-High frequency. This because we have discarded the lower-low frequency which we got in the starting case. After using all the 3 compressions we concatenate the file which we obtained into a single one. From this concatenated file we backtrack by performing anti Huffman, which is a decompression algorithm. Along with anti-Huffman we also use an inverse of LPC and backtrack to lower high signal and a higher low signal. From there we use IDWT twice and achieve to our original sampling signal. We then check the size and consistency of the output signal with the original signal that we had fed into the system.

For the above methodology the formula used for DWT is given by –

$$\phi(x) = \sum_{k=-\infty}^{\infty} a_k \phi(Sx - k).$$

For inverse DWT we use the below given formula –

C. Results

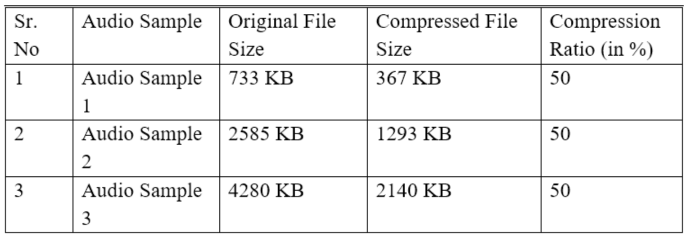
Calculating the compression ratio in audio compression is significant because it allows the user to control the amount of

$$x(t) = \frac{1}{C} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{1}{a^2} W(a,b) \Psi_{a,b}(t) \, da \, db$$

where

$$C = \int_{-\infty}^{\infty} \frac{|\Psi(w)|}{|w|} dw$$

dynamic range reduction that occurs during the compression process. A higher compression ratio means more dynamic range reduction, which can be useful in situations where the dynamic range of the original audio signal is too wide for the intended application. We have observed that the compression ratio to be near about 50 percent



Sr. No	Audio Sample	Original File Size	Compressed File Size	Compression Ratio (in %)
1	Audio Sample 1	733 KB	367 KB	50
2	Audio Sample 2	2585 KB	1293 KB	50
3	Audio Sample 3	4280 KB	2140 KB	50

Fig. 6. Results after performing audio compression using Huffman and LPC on three different audio samples

D. Conclusion

In the wavelet based method which is developed we were able to differentiate between the different types and levels of the energy which were involved. This shows the strength of the signal with regard to the signal's frequency. Lossy compressions play a very important role to compress the signals, reducing the complexity and improvising the processing speed. The preservation of quality, ability to recover original data is guaranteed by the lossless compresions. But when we apply the decompression by the inverses and we backtarck using IDWT and other fucntions we observe that a lot of noise has been added and the final output file which is obatined is about 50

ACKNOWLEDGMENT

REFERENCES

[1] L. Firmansah and E. B. Setiawan, "Data audio compression lossless FLAC format to lossy audio MP3 format with Huffman Shift Coding algorithm," 2016 4th International Conference on Information and Communication Technology (ICoICT), Bandung, Indonesia, 2016, pp. 1-5, doi: 10.1109/ICoICT.2016.7571951.

[2] M. A. Rahman, M. Jannatul Ferdous, M. M. Hossain, M. Rashedul Islam and M. Hamada, "A lossless speech signal compression technique," 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), Dhaka, Bangladesh, 2019, pp. 1-7, doi: 10.1109/ICASERT.2019.8934484.

[3] R. Pal, "Speech Compression with Wavelet Transform and Huffman Coding," 2021 International Conference on Communication information and Computing Technology (ICCICT), Mumbai, India, 2021, pp. 1-4, doi: 10.1109/ICCICT50803.2021.9510116.

[4] Y. Sharma and B. K. Singh, "Prediction of Specific Language Impairment in Children Using Speech Linear Predictive Coding Coefficients," 2020 First International Conference on Power, Control and Computing Technologies (ICPC2T), Raipur, India, 2020, pp. 305-310, doi: 10.1109/ICPC2T48082.2020.9071510.

[5] A. S. Bora et al., "Power Efficient Speaker Verification Using Linear Predictive Coding on FPGA," 2018 International CET Conference on Control, Communication, and Computing (IC4), Thiruvananthapuram, India, 2018, pp. 260-265, doi: 10.1109/CETIC4.2018.8530925.

[6] L. Laskov, V. Georgieva and K. Dimitrov, "Analysis of Pulse Code Modulation in MATLAB / Octave Environment," 2020 55th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST), Niš, Serbia, 2020, pp. 77-80, doi: 10.1109/ICEST49890.2020.9232755.

[7] A. Islam and P. Sridevi, "Pulse Code Modulations with Derivative Dependent Sampling Time Quantization and Coupled Encoding," 2019 IEEE International Conference on Signal Processing, Information, Communication and Systems (SPICSCON), Dhaka, Bangladesh, 2019, pp. 40-44, doi: 10.1109/SPICSCON48833.2019.9065159.