Graph Theory for the Discovery of Non-Parametric Audio Objects

by

Christopher Srinivasa

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Abstract

A novel framework based on cluster co-occurrence and graph theory for structure discovery is applied to audio to find new types of audio objects which enable the compression of an input signal. These new objects differ from those found in current object coding schemes as their shape is not restricted by any a priori psychoacoustic knowledge. The framework is novel from an application perspective, as it marks the first time that graph theory is applied to audio, and with regards to theoretical developments, as it involves new extensions to the areas of unsupervised learning algorithms and frequent subgraph mining methods. Tests are performed using a corpus of audio files spanning a wide range of sounds. Results show that the framework discovers new types of audio objects which yield average respective overall and relative compression gains of 15.90% and 23.53% while maintaining a very good average audio quality with imperceptible changes.

Acknowledgements

This thesis is the result of many thought-provoking discussions. For these, I would like to thank my colleagues in the Advanced Audio Systems group at the Communications Research Center of Canada, particularly Dr. Ramin Pichevar and Dr. Hossein Najaf-Zadeh. To Dr. Pichevar, thank you for your input, patience, and originality in our many mathematical conversations. To Dr. Najaf-Zadeh, your mentorship throughout my time at CRC served as the basis in forging my approach towards performing research by tackling a problem at its core rather than performing incremental contributions. Thank you as well for our many conversations where the topics varied widely from the work of this thesis to the attributes needed for a successful career in research. To my supervisor at the University of Ottawa, Dr. Martin Bouchard, thank you for being so present at times when I needed your insight and guidance both on matters related to this thesis and more generally to life as a graduate student, and at the same time for knowing exactly when to give me the intellectual freedom that I needed to find my own way. If not for these factors, I would never have grown as much as I did as both a researcher and person during my time as your student. Lastly, I would like to thank my parents Louise Cote and Rao Srinivasa for without their love and support throughout this endeavour, this thesis would not be possible. A special mention to my father who spent countless hours listening to me as I rambled on about the many ideas which I was considering and who also helped in proofreading the final work. For this, I dedicate this thesis to him.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

This thesis addresses the issue of structure. The main motivation behind structure discovery is the efficient description of data. Often times, a given amount of data contains repetitions. A simple example would be a list of numbers where a certain sequence occurs more than once. By discovering these repetitions, the list can be described more concisely than if the sequence is described as individual numbers every time it occurs. This example raises two issues about structure discovery. First, in the list of numbers, the repetitions are exact copies of a certain sequence of numbers. However, in real life settings, exact numerical repetitions are rare. In such inexact contexts, how can repetitions be identified? Secondly, humans can often easily identify the recurring sequence of numbers as they possess cognitive skills for such tasks. This is a much more difficult endeavor for a computer. As such, how can a computer find repetitions without having prior knowledge about the expected structure of the repetitions?

A prime setting where both these issues arise is in the study of audio signals. In this environment, a sound is deemed repetitive if its occurrences are acoustically equivalent to the user. The occurrences, however, may not be equivalent in their numerical descriptions. In addition, each sound may have a different structure (i.e. a different type of sound such as a harmonic or impact sound) which poses a problem in the discovery of its occurrences.

This thesis answers both above questions by introducing a novel object extraction process for object-based audio coding. The proposed method makes use of graph theory. The representation of an audio signal as a graph creates an environment where the

limitations in the field of audio compression and coding can be addressed. Although the audio coding context inspired the proposed scheme, the object extraction process is not hostage to this sole application. It is rather a graph theoretic framework whose applications are wide ranging. The interpretation of almost any problem as a graph makes this framework readily extensible. However, the aim of this thesis is to provide a proof of concept for the proposed method and the audio context is chosen as the setting to do so. As such, this thesis presents the framework within an audio context all the while placing emphasis on separating the theoretical developments from the application. The proposed scheme is thus novel from an application standpoint in that:

- it is the first time that graph theory is applied to audio

As seen throughout the thesis, the scheme is also novel in terms of theoretical developments as it involves new extensions to the areas of:

- unsupervised learning algorithms
- frequent subgraph mining methods

Section 1.2 provides an overview of the audio coding field and explains how the proposed object extraction framework addresses the pitfalls of current methods. The proposed scheme is then outlined in section 1.3.

1.2 Audio Coding

The various approaches to coding audio signals are generally classified into two categories: non-parametric transform-based methods and parametric methods.

Although lossy, non-parametric transform-based methods aim to retain the audio signal as much as possible by performing only a single time-frequency transformation of the original waveform. These methods transmit the signal as a set of time-frequency coefficients. A prime example of non-parametric transform-based coders is the MPEG-4 Advanced Audio Coder (AAC) [1]. The advantage with non-parametric transform-based methods is their ability to preserve high audio quality by capturing all the unique characteristics of the signal. However, this approach results in high data transmission rates.

In contrast to non-parametric transform-based schemes, parametric methods aim to describe a signal as a set of concise audio events using prior knowledge about the

structure of audio. It should be noted that these methods are also lossy. Parametric methods originated with the (Musical Instrument Digital Interface) MIDI format. In this format, signals are modeled and transmitted as a set of synthetic musical notes similar to a piano roll description used in sheet music. The audio events used to model signals have since evolved to yield a more flexible representation using a mixture of sines, transients, and noise. The standardized MPEG-4 Sinusoidal Coding (SSC) coder [2] and the coder presented in [3] use these three events to model a signal. The main advantage with parametric schemes is the low data rate achieved due to the concise representation. However, the main drawback is that the set of audio events available to model a signal must be defined a priori for each coder. Depending on which signal is presented to the coder, this predefined choice of events may not always be optimal to model the signal and in turn lead to a significant loss of audio quality.

Up to now, object-based schemes are viewed as an extension of parametric methods by allowing for more complex (i.e. higher level) audio events when modeling a signal. These more complex events are here onwards referred to as audio objects. Just as for parametric coders, the audio objects which are used are also developed based on prior knowledge. One such motivated object is the harmonic sound. A harmonic object is described as a combination of sines whose frequencies are evenly spaced. The object-based coders proposed in [4] and [5], and the standardized MPEG-4 Harmonic and Individual Lines plus Noise (HLIN) coder [6], make use of this object to model signals. Other than the harmonic sound, no other audio objects have been developed. This is because the acoustic community does not have enough external psychoacoustic knowledge to make additional quantitative assumptions about the structure of audio. The lack of more audio objects has made the development of object-based coders, capable of modeling a signal with objects other than harmonics sounds, rare if not inexistent.

In this thesis, a new vision for object-based audio coding is proposed. Rather than viewing this form of coding as an extension of parametric methods, as has always been done up to now, the object extraction method presented in this thesis builds on non-parametric transform-based methods. To avoid confusion between the proposed method and the current object-based methods surveyed in the previous paragraph, the latter are re-categorized as parametric object methods. In the proposed method, a time-frequency representation of the audio signal is used as the starting point for the object extraction method. A graph is then formed from the time-frequency representation. The extracted objects are combinations of coefficients which occur more than once in the time-frequency representation, here onwards referred to as *Non-Parametric Objects (NPO)*. The usage of

graph theory allows for the extraction of any recurring combination of coefficients from the time-frequency representation, thereby allowing for the extraction and successful re-synthesis of a *broader class of audio objects than those found in current parametric object-based coders*. The usage of graph theory also allows for extracted objects to be interlaced, leading to a much more advanced comprehension of structure in an audio signal. To summarize, NPOs are characterized as:

- having no predefined parameterization space meaning that the audio object itself can have any shape
- the discovery of a sound which minimizes an external cost function unrelated to the shape of the object

Figure 1.1 shows how an audio coder inspired by the proposed non-parametric object extraction method fits in amongst the other coding methods surveyed in this section.

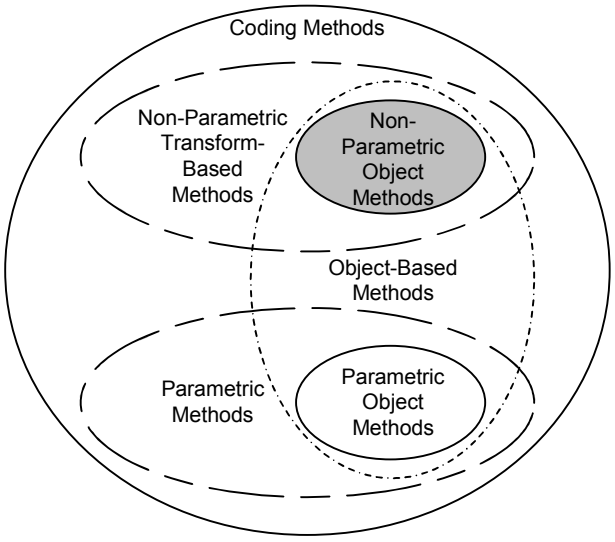


Figure 1.1: Coding Methods

1.3 Outline

Figure 1.2 shows how the proposed object extraction scheme would be used within an audio coding context. It should be noted that in this and all other block diagrams

contained in this thesis, cylinders represent storage elements, rounded rectangles represent processes, diamonds represent decision points, and parallelograms represent external functions and parameters. For their part, dashed lines represent data flow while solid lines represent process flow.

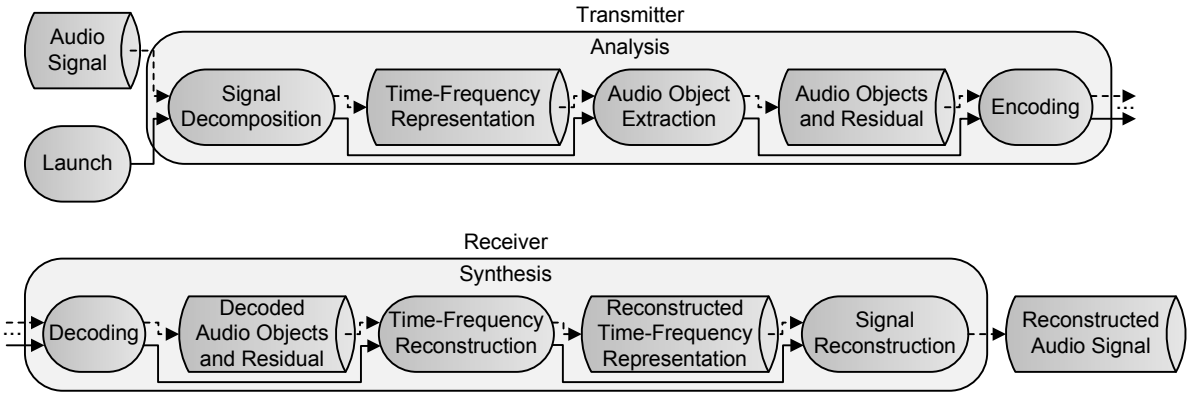


Figure 1.2: Object-Based Audio Coder

The audio coder in figure 1.2 contains the analysis part used to encode and transmit an audio segment, and the synthesis part used to receive the encoded segment from a transmission channel and convert it back to an audible format. The analysis part involves three operations: the decomposition of the audio signal into a time-frequency representation, the object extraction process, and the encoding of the objects and residual via a bit allocation scheme. The allocated bits are then sent to the receiver over a transmission channel. The synthesis part involves the decoding of the transmission channel data to obtain the various objects and residual, followed by the reconstruction of the time-frequency representation, and finally the reconstruction of the audio signal.

The focus of this thesis is on the object extraction process and not on the decomposition of the signal or the encoding of the objects. Having said this, the signal decomposition step must be addressed since it generates the required input for the object extraction process. Chapter 2 provides an overview of signal decomposition methods and provides motivation for the one used in this coder. The next three chapters contain the core material of this thesis since they pertain to the object extraction process. Chapter 3 contains a review of all object and structure discovery methods and then outlines a graph theory motivated extraction scheme. Chapter 4 discusses the interpretation of the problem as a graph using learning algorithms. Chapter 5 then shows how repetitive structures can be extracted from the graph. Chapter 6 addresses the recording of the

extracted objects. Chapter 7 discusses the reconstruction of the time-frequency representation based on the extracted objects and residual followed by the reconstruction of the audio signal. In chapter 8, an experiment making use of the proposed framework is constructed. The performance of this experiment on a database of audio signals is then reported in chapter 9. A summary of the thesis is provided in chapter 10. Lastly, as the proposed scheme is the first of its kind, much room exists for further improvements both at the theoretical framework level and within the context of audio coding. These potential improvements are discussed in chapter 11.

Chapter 2

Signal Decomposition

2.1 Introduction

When analyzing an audio signal, it is desirable to interpret it in a manner similar to that of the human biological system. Using such bio-inspired knowledge increases the likelihood that the observations made when analyzing the signal correspond well with what actually occurs in the human brain. The most basic and common representation for an audio signal is as a waveform $\mathbf{x}$, shown in figure 2.1, sampled at a particular frequency F_s to yield N samples. However, the time domain by itself fails to illustrate how the

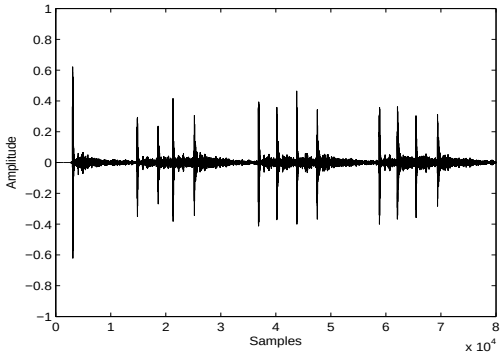


Figure 2.1: Audio Signal Waveform

human ear processes sound. It is widely known in the field of psychoacoustics that the inner ear contains hair cells which fire signals to the brain only when the membrane on which they sit oscillates at certain frequencies. This allows the ear to decipher an audio

signal based on its frequency content.

Within an engineering context, this psychoacoustic knowledge motivates the time-frequency decomposition of an audio signal. To achieve such a representation, the audio signal is evaluated against a set of kernels which represent different frequencies. Its time-frequency representation is then determined based on how much it correlates with the different kernels. The shape and properties of the kernels to be used for the decomposition as well as the different available decomposition algorithms are two well documented topics. In this chapter, both subjects are explored and motivation is provided for the choice of Gammatones for the kernels and a variant of Matching Pursuit for the decomposition algorithm, as shown in figure 2.2.

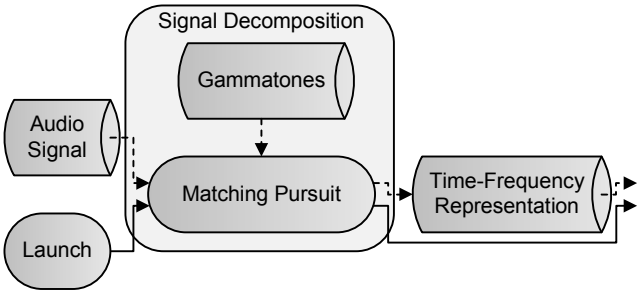


Figure 2.2: Signal Decomposition Block

2.2 Kernels

Kernels are a set of predefined finite length functions. Many types of kernels sets exist and each set is motivated by its own psychoacoustic and signal processing literature.

In most sets, each kernel $\mathbf{g}$ represents a certain center frequency channel h. However, not all sets abide by this rule of thumb. In this section, both types of kernels sets are examined. When kernels in a set are assigned respective channels, each kernel can be mathematically expressed in the time domain as a carrier signal waveform $\mathbf{u}$ modulated by a finite length window $\mathbf{w}$ as shown in equation 2.1.

$$\mathbf{g}_h = \mathbf{w}_h \mathbf{u}_h \tag{2.1}$$

Each kernel is then normalized to have unit energy. For these types of kernel sets, section 2.2.1 discusses the number of frequency channels $|H|$ and their distribution along

the frequency axis, section 2.2.2 details the attributes of the carrier signal, and section 2.2.3 presents different windows which give rise to special cases of kernels. Lastly, section 2.2.4 provides an overview of those types of kernels which cannot be described as per equation 2.1.

2.2.1 Frequency Distribution

The first factor which affects the frequency axis is the number of frequency channels $|H|$ (i.e. the number of kernels) in the kernel set. This number determines the resolution of the frequency axis in the time-frequency decomposition.

The second factor of interest is the distribution of the center frequencies. Two types dominate: the uniform distribution and the Equivalent Rectangular Bandwidth (ERB) distribution. The uniform distribution has been extensively used in signal analysis. Here, the center frequencies are spaced out evenly across the frequency range. Although, this frequency distribution is simple, its main drawback is that it does not take into account any psychoacoustic knowledge. In contrast, the ERB distribution is determined from strict psychoacoustic experiments [7]. It best approximates the behavior of the human hearing system since it mimics the resolution of hair cells along the cochlea inside the human ear [7]. In [7], it is shown that human hearing can discriminate better amongst low frequencies than high frequencies. As such, the ERB distribution skews the positioning of the center frequencies towards the low end of the frequency axis. Figure 2.3 shows center frequencies according to the ERB distribution.

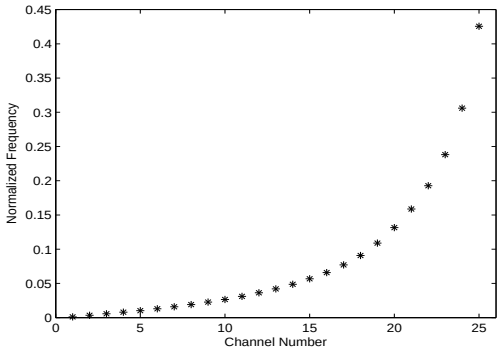


Figure 2.3: Center Frequencies According to ERB Distribution for $|H| = 25$

2.2.2 Carrier Signal

A carrier signal is described as a complex sinusoid, as shown in equation 2.2.

$$u_h[n] = \exp^{-\jmath 2\pi f_h n} = \cos(2\pi f_h n) - \jmath \sin(2\pi f_h n) \tag{2.2}$$

Note that for a complex sinusoid, the real part - a cosine function - and the imaginary part - a sine function - are orthogonal. Although most kernel sets utilize the entire complex sinusoid, some elect to only make use of its real part, as shown in equation 2.3.

$$u_h[n] = \Re\left\{\exp^{-\jmath 2\pi f_h n}\right\} = \cos(2\pi f_h n) \tag{2.3}$$

This point is further discussed in the algorithms of section 2.3.

2.2.2.1 Frequency Modulation

The carrier signal can sometimes be subject to frequency modulation. This is done by adding an additional parameter known as the modulation index c in the expression for the carrier signal. One such example is shown in equation 2.4.

$$u_h[n] = \cos(2\pi f_h n + c \ln(n/F_s)) \tag{2.4}$$

The modulation index can be restricted to taking on a predefined finite set of values, as done for the center frequencies, or it can be found using an optimization procedure during the decomposition process. Figure 2.4 shows the waveform of a carrier signal subject to frequency modulation as per equation 2.4. In this figure, the frequency modulation is

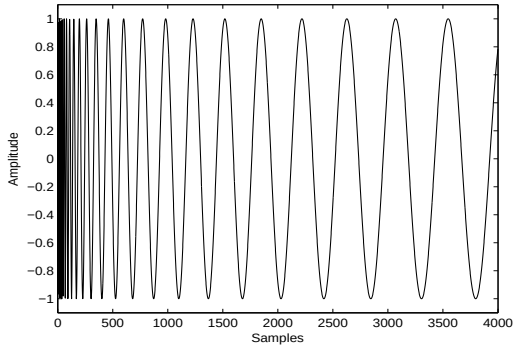


Figure 2.4: Waveform of Carrier Signal with $c = 20$

clearly visible since the period of the carrier signal decreases as time progresses.

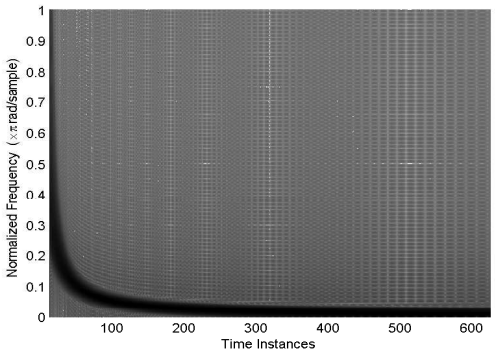


Figure 2.5: Time-Frequency Behaviour of Carrier Signal with $c = 100$

Figure 2.5 shows the time-frequency behaviour of a frequency modulated carrier signal obtained by using equation 2.4. Due to the logarithmic term associated with the modulation index, the curve observed in figure 2.5 is similar to that of a $1/n$ function [8]. This is because the curve is determined by the rate of change of the modulation function as shown in equation 2.5. For a negative modulation index, the corresponding time-frequency curve is directed upwards.

$$\frac{d}{dn}\left\{c\ln\left(n/F_s\right)\right\} \quad = \quad cF_s/n \tag{2.5}$$

2.2.3 Windows

Many kernel sets differentiate themselves by the shape of their finite length modulating window. Some commonly used windows in signal analysis are the rectangular, Hanning, and Hamming windows each described in equations 2.6, 2.7, and 2.8 respectively [9]. Figure 2.6 shows all three windows in the time domain.

$$w_h[n] \quad = \quad \begin{cases} 1, & 0 \leq n \leq M-1 \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

$$w_h[n] \quad = \quad \begin{cases} 0.5 - 0.5\cos\left(2\pi n/M\right), & 0 \leq n \leq M-1 \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

$$w_h[n] \quad = \quad \begin{cases} 0.54 - 0.46\cos\left(2\pi n/M\right), & 0 \leq n \leq M-1 \\ 0, & \text{otherwise} \end{cases} \tag{2.8}$$

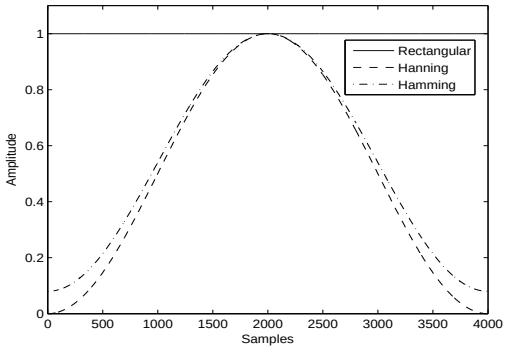


Figure 2.6: Rectangular, Hanning, and Hamming Windows

The use of any of the above windows combined with complex carrier signals whose frequency channels are uniformly distributed gives rise to the kernel set used in Fourier analysis. Fourier analysis itself is described in section 2.3.1. This is the most commonly used kernel set in time-frequency representations due to its favorable mathematical properties. First, due to the symmetric nature of the windows in the time domain, the frequency domain dual of the kernels exhibit unique characteristics which can minimize the number of computations required when decomposing an audio signal. Second, it can be shown that any two kernels with different frequencies from this set are mutually exclusive, thereby forming an orthogonal set [9].

In signal processing, it is a well known fact that a trade off exists between time and frequency resolution based on the window length [9]. A large window length which leads to a high frequency resolution results in poor time resolution and vice-versa. This trade off is not argued here. It is rather the fact that constant-length rectangular, Hanning, and Hamming windows impose the same time-frequency trade off characteristics for all kernels in a kernel set which disadvantages these window types. This is because the length of these windows is fixed regardless of the frequency of the carrier signal which sits inside them. To be appropriately modelled, every kernel in a kernel set must contain at least one period of its carrier signal. This period is longest for the carrier signal whose frequency is lowest. As such, the minimum length of the window for all kernels in a set is bounded by the kernel with the longest period. However, kernels with higher frequency carrier signals can afford a shorter window since the length of their period is much shorter. The fact that the window length cannot be adjusted for higher frequency

kernels compromises their time resolution. This is because these kernels contain more than one period within the window and it cannot be said which of these cycles correlates best with the signal during its decomposition. Hence, the time positioning of the kernels in the time-frequency representation becomes coarse.

A second disadvantage with rectangular, Hanning, and Hamming windows is that they are not bio-inspired. There is no evidence to suggest that the inner ear decomposes a signal using kernels modulated by these windows. From an engineering standpoint, it has also never mathematically been shown that kernels resulting from these windows are the best choice to represent audio signals.

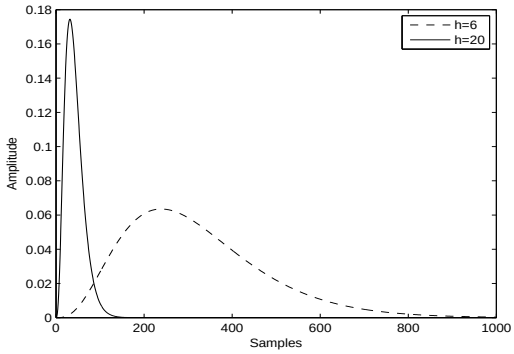
In contrast to Fourier kernels, Gammatones are a set of psychoacoutically motivated kernels which were introduced in [10]. The Gammatone shape was mathematically confirmed in [11] where a series of natural sounds were presented to an undefined set of kernels whose waveform shapes were allowed to fluctuate. The goal was to adaptively learn the shape of these kernels which would best model the natural sounds. The resulting kernel waveform shapes were those of the Gammatone kernel set. Gammachirp kernels, introduced in [8], represent an extension of Gammatone kernels when frequency modulation is allowed for the carrier signal.

The window for a Gammatone kernel is defined as per equation 2.9.

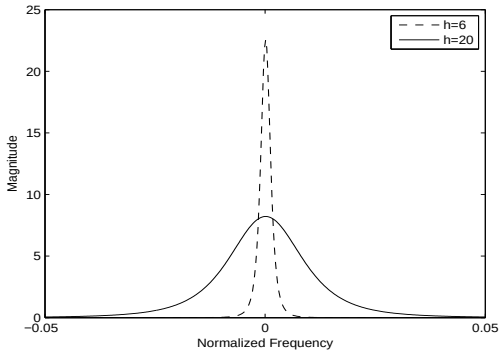
$$w_h[n] = \begin{cases} (n/F_s)^3 \exp^{-2\pi erb(f_h)n}, & 0 \leq n \leq M-1 \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

The Gammatone kernel set is obtained using this window combined with carrier signals whose frequency channels are distributed according to the ERB distribution. In contrast to the Fourier kernel set, Gammatone kernels do not form an orthogonal set.

The key advantage with Gammatone kernels is that the shape of the window is dependent on the frequency of the carrier signal which lies inside it, as shown in figure 2.7. As the frequency of the carrier signal increases, the effective length M_{eff} of the window in the time domain shortens which is not the case for the windows used in Fourier analysis. This resolves the time positioning issue described earlier for rectangular, Hanning, and Hamming windows. Naturally, shorter effective window lengths - and in turn higher time resolution - at high frequencies translates into lower frequency resolution for these same kernels, as per time-frequency trade off properties. This means that the magnitude response of high frequency kernels looses precision by covering a broader range of frequencies, as shown in figure 2.7. However, this behaviour is desired as it mimics that of the inner ear, as described earlier in section 2.2.1. The key point is that each kernel



(a) Gammatone Windows for $h = 6$ and $h = 20$, $|H| = 25$



(b) Magnitude of Frequency Response for Gammatone Windows for $h = 5$ and $h = 19$, $|H| = 25$

Figure 2.7: Gammatone Windows and Frequency Responses for $|H| = 25$

in a Gammatone kernel set can have unique time-frequency trade off characteristics due to the variable effective window length as opposed to Fourier kernels where all kernels in a set are bound to the same time-frequency trade off characteristics due to the fixed window length of rectangular, Hanning, and Hamming windows.

The rate at which the Gammatone window shortens is governed by the ERB function in equation 2.9. The ERB function determines the appropriate frequency coverage, also known as the bandwidth, for each kernel based on the frequency of its carrier signal. For a complete Gammatone kernel set with ERB distributed carrier signal frequencies, the use of the ERB function ensures that no frequency range is left uncovered in the frequency domain, as shown in figure 2.8.

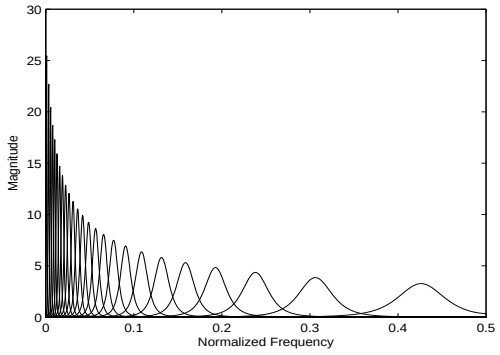


Figure 2.8: Magnitudes of Frequency Responses for Gammatone Set with $|H| = 25$

The mathematical confirmation that Gammatone kernels are the best to model a wide range of sounds combined with the fact they are psychoacoustically motivated is what make this kernel set the preferred choice for the signal decomposition process.

2.2.4 Other Kernels Sets

In addition to the kernel sets presented in the previous sections, many other sets exist which cannot be parametrized using a carrier signal and a window. Similarly to what was done using natural sounds to confirm the shape of the Gammatone kernel set, the kernels for these sets were determined using learning algorithms to find the waveforms which would best model a specific type of sound in each case. Three such types of kernel sets are the speech set, the click set, and the white noise set. The reader is referred to [12] for visual depictions of all three types of sets. The speech set contains kernels which are designed to model specific speaker characteristics such as unvoiced sections when pronouncing fricatives and voiced sections when uttering vowels. The click set is designed to model events with sharp onsets and short durations due to the spike-like shape of the kernels belonging to the set in question. Lastly, the white noise set handles audio events which are best described by chaos or randomness as denoted by the lack of structure in the kernel waveforms for this set.

The main drawback of all the kernel sets presented in this section is the restrained applicability of the respective sets to model only specific sounds.

2.3 Algorithms

Time-frequency representations aim to describe the variations in the frequency content of an audio signal as it evolves temporally. To this effect, section 2.3.1 presents the underlying principle used to achieve time-frequency transformations. The section also progressively shows how, when combined with a particular kernel set presented in section 2.2, this principle gives rise to the most commonly used transformation in the field of signal processing.

Section 2.3.2 then presents alternate forms of time-frequency transformations, known as sparse decompositions. These types of transformations are based on a slight modification to the underlying principle used in section 2.3.1. The motivation for the usage of sparse decompositions as well as some pitfalls of these representations are presented.

2.3.1 Transforms

The goal of all time-frequency transformations is to decompose an audio signal based on how well it matches with a given kernel set. The extent to which the signal matches with a particular kernel can be described mathematically as the inner product. In most cases, the kernel is much shorter than the audio signal. As such, a time instance p along the audio signal at which the inner product is taken must be defined. This time instance denotes the spot along the audio waveform where the start of the kernel waveform is placed. Due to the finite length of the kernel window, only the M samples starting from this time instance onwards are used from the signal for the inner product. Mathematically, the above observations made for the inner product of an audio signal at a particular time instance with all kernels from a kernel set translate to equation 2.10.

$$X_p(h) = \sum_{n=0}^{M-1} x[n+p]g_h[n] = \sum_{n=0}^{M-1} x_p[n]g_h[n] \qquad \forall h \in H \tag{2.10}$$

When evaluated with a rectangular windowed Fourier kernel set, this equation corresponds to the definition of the Discrete Fourier Transform (DFT) as shown in equation 2.11 [9].

$$X_p(h) = \sum_{n=0}^{M-1} x_p[n] \exp^{-\jmath 2\pi f_h n} \qquad \forall h \in H \tag{2.11}$$

If equation 2.10 is evaluated with a kernel whose carrier signal is complex, the result - from here onwards referred to as the coefficient - is also complex. The real part of the coefficient indicates how much the audio signal matches with the cosine part of the carrier signal whereas the imaginary part indicates how much the audio signal matches with the sine part. However, the complex coefficient may indicate that the audio signal matches best with a kernel whose carrier signal is an intermediate of the cosine and sine parts. This can be better illustrated by viewing the complex coefficient as a phasor, as shown in figure 2.9. In this figure, the length of the projected phasor onto the real and imaginary axis represent the real and imaginary parts of the coefficient. The length of the actual phasor represents the magnitude of the coefficient and its angle with respect to the real axis shows its phase. The phase represents the amount of slide required for a carrier signal within the window of a kernel to maximize the magnitude of the coefficient. As this parameter is periodic, its value resides between 0 and 2π.

If the carrier signal of the kernel in equation 2.10 is purely real, the coefficient is a signed real number whose absolute value is its magnitude and whose sign is its phase.

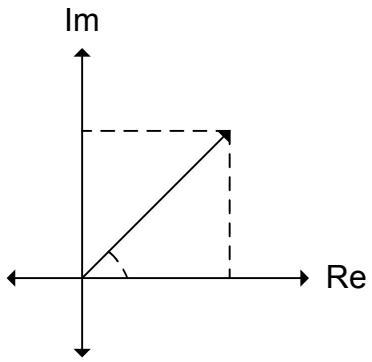


Figure 2.9: Complex Phasor

This can be alternatively viewed as the phasor in figure 2.9 being restricted to the real axis with possible phases of 0 (i.e. a positive sign for the coefficient) or π (i.e. a negative sign for the coefficient). As such, utilizing a purely real carrier signal simplifies the requirement of recording the phase, when computing the inner product, since only a binary value needs to be recorded.

In summary, taking the inner product of an audio signal requires the recording of four values: the frequency channel of the kernel h, the time instance along the audio signal p, the magnitude of the coefficient $\|X_p(h)\|$, and the phase of the coefficient $\angle X_p(h)$.

To get a good understanding of the temporal evolution of an audio signal, the inner product must be taken at many time instances. At each time instance, the inner product is taken between the signal and all the kernels of the kernel set. The above steps, extend the definition of the DFT mentioned earlier to the definition of the Short Time Fourier Transform (STFT) [9], as shown in equation 2.12.

$$X_p(h) = \sum_{n=0}^{M-1} x_p[n] \exp^{-\jmath 2\pi f_h n} \qquad \forall h \in H; \forall p \in \{0, \ldots, N\} \tag{2.12}$$

Within the context of the STFT however, often times the rectangular window used in the Fourier kernel set is replaced by a Hamming or Hanning window to allow for smooth transitions from one time instance to the next in the signal analysis. The interval between the time instances at which the inner products are taken is called the hop size N_{hop}. It should be noted that the hop size may be less than the length of the kernels thereby causing some overlap of the audio sections being used for the inner products. This simply means that some of the audio samples used for one inner product may be reused for the next.

The output of time-frequency transformations is best viewed in a two dimensional map where the vertical axis represents frequency and the horizontal axis represents time. All coefficients are placed as pixels in this map by finding the time instance p at which their inner product is taken along the horizontal axis and the frequency channel h of the kernel involved in their inner product along the vertical axis. The magnitudes of the coefficients are illustrated via the darkness levels of the pixels in the map. It should be noted, that the phases of the coefficients are not represented in such a map.

Figure 2.10 illustrates the map just described containing the result of an STFT transform applied to the waveform of figure 2.1. In the case of this transform, the map is also called a spectrogram.

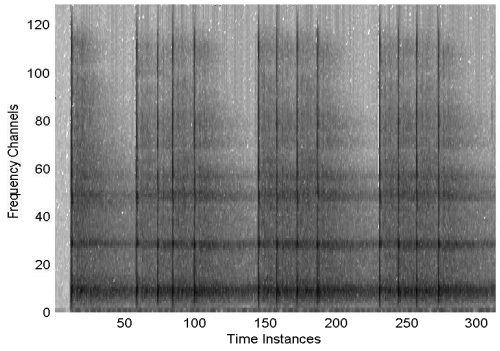


Figure 2.10: Spectrogram with $M = 256$, $N_{hop} = 128$, and $|H| = 128$

2.3.2 Sparse Decompositions

In the transforms presented in section 2.3.1, coefficients are calculated for every time instance and with all kernels from the kernel set. This leads to a dense time-frequency representation of a signal with many non-zero coefficients. In contrast to these transforms, sparse decompositions aim to produce a time-frequency representation of an audio signal with the smallest number of non-zero coefficients. The biological motivation behind these decompositions is that the neural response in humans to external stimulus is sparse [13]. In other words, when presented with a sound, very few of the many neurons connected to the hair cells in the inner ear fire.

The challenge is now to come up with the best time-frequency representation for a signal all the while making sure that it remains sparse. Many proposed approaches

formulate the above statement as a constrained optimization problem and attempt to solve it using interior point methods or gradient algorithms. The approach proposed in [14] is of particular interest because it relies heavily on a biological phenomenon which occurs amongst neurons when responding to a stimulus. The biological concept states that neurons in the biological system can, not only output positive electrical discharges, but can also output negative discharges known as inhibitions. When one neuron fires a positive electrical discharge in response to a stimulus, it also fires inhibitions which can cancel out the positive discharges of other neighboring neurons essentially making them zero. The author in [14] artificially reproduced this phenomenon to guarantee the sparseness of the final time-frequency representation.

One common method originally introduced in [15] to yield sparse decompositions is called Matching Pursuit (MP). MP produces a sparse time-frequency representation of an audio signal through the iterative procedure outlined in figure 2.11. Because of its

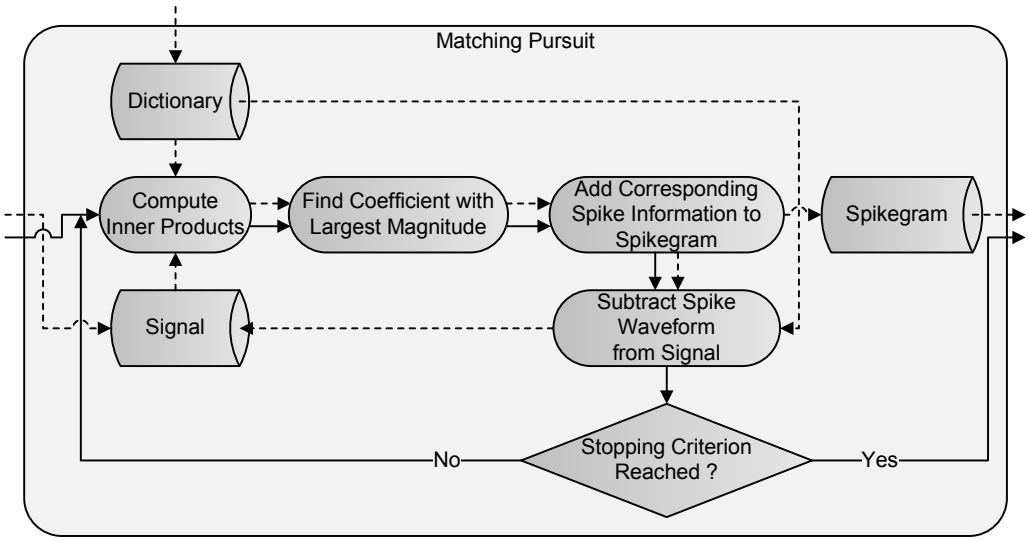


Figure 2.11: Matching Pursuit Block

sparse nature and biological motivation based on the spike-like firing of neurons, the time-frequency representation resulting from the MP algorithm is known as a spikegram and each non-zero coefficient in the representation is known as a spike. In the audio context, the dictionary used by MP is the preferred kernel set amongst those presented in section 2.2. The operation of MP is as follows. Starting with the original signal, the inner product is taken at all time instances specified by the hop size with all kernels from the kernel set. The time instance and frequency channel of the kernel yielding the

coefficient with the largest magnitude are recorded. These two coordinates along with the magnitude and phase of the coefficient form the spike $\mathbf{s} = \langle s_{mag}, s_{ang}, s_{pos}, s_{chan} \rangle$ to be extracted, as shown in equations 2.13 to 2.16.

$$s_{mag} = \max \left\{ \| \sum_{n=0}^{M-1} x_p[n] g_h[n] \| \right\} \qquad \forall h \in H; \forall p \in \{0, \ldots, N\} \tag{2.13}$$

$$s_{ang} = \arg\max_{\angle} \left\{ \| \sum_{n=0}^{M-1} x_p[n] g_h[n] \| \right\} \qquad \forall h \in H; \forall p \in \{0, \ldots, N\} \tag{2.14}$$

$$s_{pos} = \arg\max_{p} \left\{ \| \sum_{n=0}^{M-1} x_p[n] g_h[n] \| \right\} \qquad \forall h \in H; \forall p \in \{0, \ldots, N\} \tag{2.15}$$

$$s_{chan} = \arg\max_{h} \left\{ \| \sum_{n=0}^{M-1} x_p[n] g_h[n] \| \right\} \qquad \forall h \in H; \forall p \in \{0, \ldots, N\} \tag{2.16}$$

The spike is added as the first spike (i.e. $i = 1$) to the spikegram and extracted from the original signal (i.e. $\mathbf{x}^1 = \mathbf{x}$) by multiplying the kernel waveform corresponding to the appropriate frequency channel with the coefficient and subtracting it from the waveform of the signal at the appropriate time instance, as shown in equation 2.17. In equation 2.17, it is assumed that kernels with purely real carrier signals are employed thereby restricting the phase of the spike to either 0 or π.

$$x^{i+1}[n] = \begin{cases} x^i[n] - s^i_{mag} \exp^{\jmath s^i_{ang}} g_{s^i_{chan}}[n - s^i_{pos}], & s^i_{pos} \leq n \leq s^i_{pos} + M - 1 \\ x^i[n], & \text{otherwise} \end{cases} \tag{2.17}$$

Note that this subtraction makes the spike orthogonal to the residual. The residual signal is then returned as the input signal for the next (i.e. $i + 1$) MP iteration. This process repeats until a desired number of spikes $|S|$ are extracted or until a certain amount of energy is removed from the original signal. It should be noted that once the iterative process terminates, the residual signal is normally discarded and only the extracted spikes are used to represent the signal. The computational details of MP are further discussed in [16]. Figure 2.12 shows the spikegram for the audio signal of figure 2.1. It should be noted that the magnitude and phase of the spikes are not shown in this figure.

A key problem with MP is the presence of spikes in the spikegram to compensate for other spikes. When extracting a spike at a particular MP iteration, the spike is almost

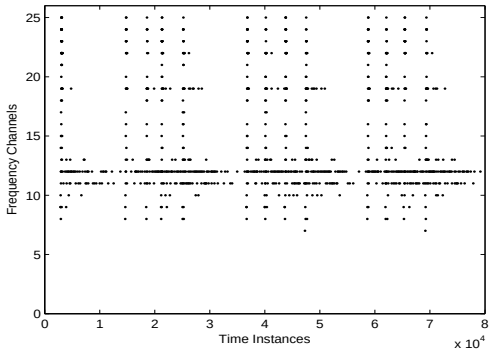


Figure 2.12: Spikegram with with $M = 4000$, $N_{hop} = 1$, $|H| = 25$, and $|S| = 5000$

never a perfect match with the signal. As such, its subtraction from the signal creates constructive interference and introduces artifacts into the residual signal. It often occurs that the spike extracted at the next MP iteration tries to remove these artifacts. This essentially means that a spike is placed in the spikegram, not to extract any energy from the original signal, but rather solely to compensate for the artifacts of another extracted spike. To address this problem, Orthogonal Matching Pursuit (OMP) was introduced in [17]. In OMP, each spike which is extracted must not only be orthogonal to the residual signal after its extraction but also orthogonal to all the spikes which are extracted before it. This guarantees that a spike is not present to compensate for the artifacts introduced into the residual signal by another extracted spike. The problem with OMP, however, is that it is very time consuming since every time a spike is extracted, the positioning of all spikes which were extracted before it must be adjusted to make sure that orthogonality is preserved amongst all of the spikes.

Despite its previously above mentioned disadvantage, the main drawback of MP is also its operational speed due to the many inner products to be taken and the extensive search time to find the coefficient with the largest magnitude at each iteration. However, efforts are currently being placed on addressing this issue by presenting methods to improve the tractability of the algorithm such as those in [18] and [16]. These improvements make MP one of the fastest methods amongst the family of sparse decomposition methods, much faster than OMP.

Another variant of MP, introduced in [19] is known as Perceptual Matching Pursuit (PMP). In PMP, psychoacoustic masking effects are taken into account during the iterative signal decomposition process. The reader is referred to [19] for the exact speci-

fications of the employed masking model. PMP operates exactly like MP apart from one modification: at each iteration, coefficients whose values fall below the mask created by the previously extracted spikes (i.e. coefficients which are inaudible) are set to zero and therefore cannot become the extracted spike for the iteration in question. Preventing inaudible coefficients from becoming spikes translates into an even more sparse spikegram and the modification from MP mentioned above is very minor. These two factors thus make PMP the preferred choice for the signal decomposition algorithm in this thesis.

One must be careful of the choice of carrier signal for the kernels when using MP or any of its variants (e.g. PMP). Because the method is iterative, it chooses the kernel whose coefficient magnitude is largest as the next spike to be extracted. This choice may be different depending on whether the kernels have complex or purely real carrier signals. Figure 2.13 illustrates an example where this could occur. In this example, the

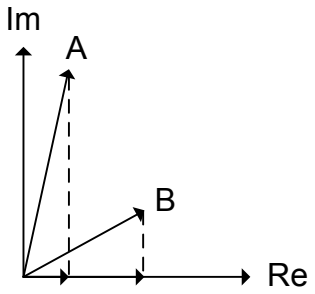


Figure 2.13: Vector Projection

MP-based algorithm must choose between two possible coefficients - 'A' or 'B' - as the next spike to be extracted. If kernels with complex carrier signals are used to compute the inner products, the magnitude of the coefficients for 'A' and 'B' are illustrated by the length of the angled vectors in the figure since the coefficients are complex. In this case, the MP-based algorithm would pick 'A' as the next spike to be extracted since the length of its vector is longer than that of 'B'. However, if kernels with purely real carrier signals are used to compute the inner products, the magnitude of the coefficients for 'A' and 'B' are illustrated by the length of the projection of the vectors onto the real axis since the coefficients are real. In this second case, the MP-based algorithm would select 'B' as the next spike to be extracted since the length of its projection is greater than that of 'A'. Since the MP-based algorithm operates with the residual signal from one iteration as the input to the next iteration, this varied choice of coefficient at one iteration affects the spikes extracted in the next iterations and in turn the final spikegram. Having said this,

experiments regarding the above-illustrated issue were performed in [16] and results show that the usage of kernels with purely real carrier signals as opposed to complex carrier signals do not affect the final time-frequency representation very much. For this reason, Gammatones with purely real carrier signals are used in this thesis since the phase of the coefficients resulting from these kernels is much easier to record than that resulting from kernels with complex carrier signals, as previously discussed in section 2.3.1.

Chapter 3

Audio Object Extraction

3.1 Introduction

The proposed object extraction framework is the core of this thesis. In this chapter, a graph theoretic approach is presented to extract audio objects from the time-frequency representation produced by the signal decomposition step.

Many methods have already been developed to find parametric audio objects. In [20], a Bayesian model is designed to infer harmonic parameters from a time-frequency representation of an audio signal. Statistical inference is also used to model audio events in [21] and [22]. The works in [23] and [24] propose a different approach by integrating the object extraction process inside the signal decomposition step and imposing constraints as to how the signal can be decomposed. Once again, however, this scheme can only extract parametric objects such as harmonics or transients. Other approaches to model transient audio events are presented in [25] and surveyed in [26].

As mentioned in chapter 1, the object extraction method proposed in this thesis aims to find objects free of a predefined parametrization space. In this spirit, emphasis is placed on reviewing other object extraction methods capable of discovering these kinds of objects as opposed to parametric objects. The advantage with these methods is that the shapes of the discovered objects are completely unrestricted. This allows for any type of object to be discovered. Section 3.2 contains a detailed survey of such methods. The proposed graph theoretic approach for object extraction is then presented in section 3.3.

3.2 Current Object Extraction Methods

The methods surveyed in this section aim to find structure in a matrix. Within the context of audio, the time-frequency representation resulting from the signal decomposition process is this matrix. However, these methods can be used for applications other than audio. To avoid loss of generality, the notion of a matrix is used rather than a time-frequency representation. It should also be noted that the methods can all be readily extended to higher dimension data sets.

The first method, originally presented in [27], is known as Non-negative Matrix Factorization (NMF). In NMF, the input matrix is factorized as the product of two matrices based on a predefined number of desired objects. The first matrix models the vertical behaviour of the discovered objects. As such, it has a number of rows equal to that of the input matrix and a number of columns equal to the number of desired objects. The other matrix is responsible for the horizontal behaviour of the discovered objects and thus has a number of columns equal to that of the input matrix and a number of rows equal to the number of desired objects. The resulting objects are obtained using a column from the first matrix and a row from the second matrix (i.e. the first column of the first matrix and the first row of the second matrix represent the first object, the second column of the first matrix and the second row of the second matrix represent the second object, etc.). The factorization of the original input matrix is an iterative process. At each iteration, new versions of the factored matrices are obtained using an update equation based on the previous versions of the factored matrices and the original input matrix. This type of update is similar to that done in many optimization procedures. The only constraint in NMF is that all coefficients in the factored matrices must be non-negative. Convolutional variants of NMF were introduced in [28] and [29] to help the discovery of objects which are not positioned strictly horizontally or vertically in the original input matrix. These methods use several pairs of factorization matrices. Each pair captures a horizontally or vertically shifted version of the objects. In [30], a sparseness constrained version of convolutional NMF was introduced to minimize the number of non-zero coefficients in the factorization matrices. This helps to localize the position and better define the shape of the discovered objects. NMF and its variants have recently been used on spectrograms in [31] to find audio objects in the context of auditory scene analysis. A disadvantage with NMF methods is that the number of objects and the size of the objects are restricted by the size and number of pairs of factorization matrices which must be defined prior to factoring the input matrix. Choosing these parameters a priori requires external

knowledge about the application in which NMF is being used. An incorrect selection could lead to incoherent objects. The main drawback with NMF methods, however, is that they do not allow for a residual to exist in the matrix. In other words, the methods assume that the entire input matrix can be factored into objects. There may be some areas of the input matrix which do not exhibit any structure. The NMF methods should be capable of identifying and considering these areas as residual since this may be difficult to do manually. Instead, these areas are included in the factorization process and inadvertently affect the shape of the discovered objects.

Parallel Region Growing (PRG), developed in [32], is another method for structure discovery often used in computer vision for image segmentation and object recognition. In PRG, the objects are grown iteratively. First, each coefficient in the matrix is initialized as being its own object. An application-specific similarity measurement is then performed between all spatially adjacent objects in the matrix. The pairs of objects whose similarity score falls within a threshold of the maximum score are merged together to form bigger objects. This process repeats by taking similarity measurements based on the new objects and checking for further mergers. The main drawback with PRG is the spatial constraint when growing the objects. Only spatially adjacent objects in the matrix can be checked for mergers. There may be objects in the matrix which would not be discovered because they are not contiguous.

The Pattern-Based Representation (PBR) introduced in [33] evolves from the field of computer performance to efficiently represent sparse matrices. It is presented as an alternative to the Compressed Sparse Row (CSR) representation, currently used in many softwares, as the indexation scheme requires less memory access time. However, it is presented in this thesis purely for its structure discovery benefits. In PBR, the input matrix is partitioned into square patches (i.e. smaller matrices) of a predefined size. The patches which are identical are identified as instances of the same object. In addition to having to define in advance the size of the patches, the PBR partitioning scheme simply assumes that the positioning of the partitions starts in the top-left corner of the matrix and progresses contiguously at intervals equal to the size of the patches going downwards and rightwards to cover the matrix in a grid like fashion. This is problematic as it may not be the optimal positioning to find the maximum number of identical patches and maximize the number of discovered objects. On the positive side, however, the PBR representation allows for a residual. Even if patches do not occur more than once, they are still represented as a single instance of an object. This way the entire matrix is represented.

3.3 Graph Theoretic Approach to Object Extraction

Figure 3.1 summarizes the proposed object extraction scheme. The proposed method

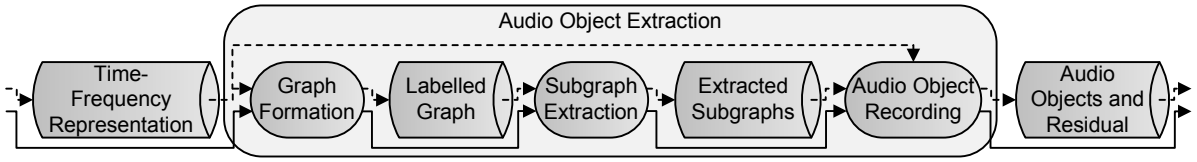


Figure 3.1: Audio Object Extraction Block

is loosely based on the one used for image analysis in [34]. The spikegram, which is the time-frequency representation used in this thesis, is first converted to a graph G composed of a set of vertices V and a set of edges E. Each spike is represented as a vertex v. As a convention, the vertices are indexed going from left to right based on the temporal position of the spikes in the spikegram, as shown in figure 3.2. The relationships

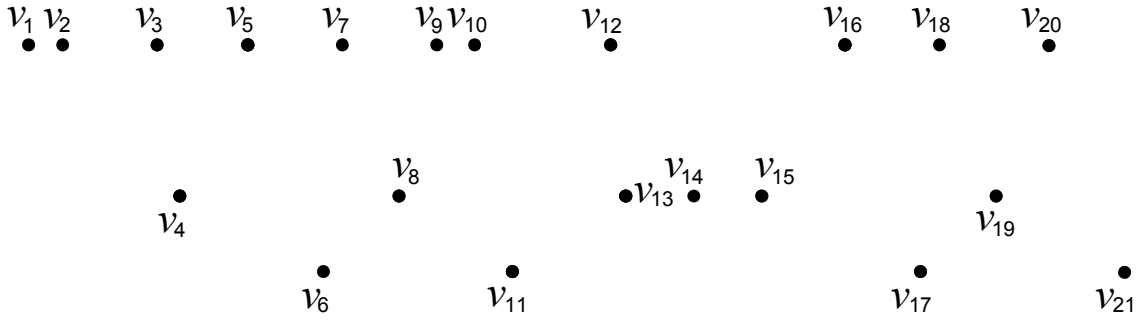


Figure 3.2: Vertices

between the vertices are represented by placing edges e in the graph as shown in figure 3.3. It should be noted that, unless otherwise mentioned and although not done in figure 3.3 for ease of viewing purposes, edges are placed between any two vertices in the graph such that every vertex is directly linked to every other vertex. This is also known as a complete graph. By doing so, the resulting graph represents the entire solution space also known as the set of all possible solutions. This means that if a particular object exists in the spikegram, it is guaranteed to be hidden within the graph. In turn, it allows to find the objects as a subset of the solution space.

Contrary to the image analysis method in [34] and the object extraction methods surveyed in section 3.2, the proposed scheme finds objects by examining both the similarity

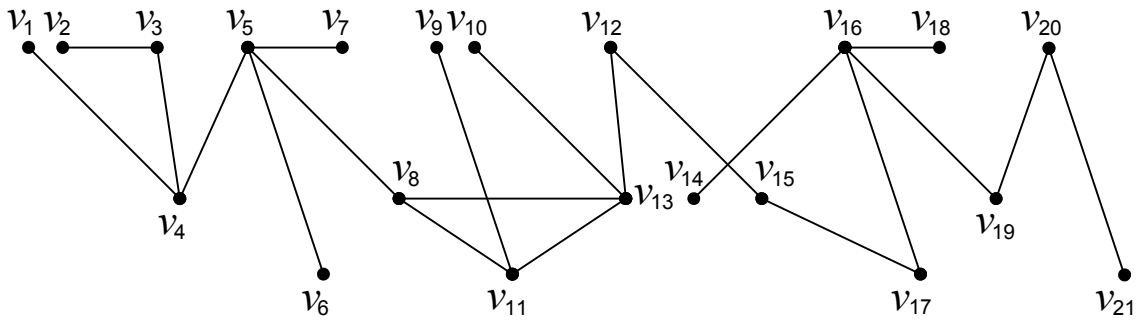


Figure 3.3: Unlabelled Graph

of the vertices themselves and the similarity of the relationships between the vertices, rather than only based on the former. This gives the proposed scheme greater flexibility over its counterparts. It is done by working with the edges in the graph rather than with the vertices since the edges are capable of containing both pieces of information (i.e. information about vertices and relationships between vertices). As such, the edges in the graph are assigned labels l similarly to what is done in [35]. Edges which are deemed similar are assigned the same label. Each edge is characterized based on the parameters of the two spikes associated with the vertices which it links. Since these vertices are common to more than one edge in the graph, the spikes associated with these vertices are reconsidered for each of the edges. By doing so, each edge is considered as being standalone even though the vertices which it links forcibly belong to other edges as well. This is the motivation behind labelling edges as opposed to vertices. The label assigned to each edge thus only implies knowledge about the two vertices when utilized to form the edge in question. It does not say anything about either of the two vertices when these are used within the context of other edges.

Figure 3.4 shows the graph of figure 3.3 once labelled. The labelling step is performed using a learning algorithm. Since the audio signal is reconstructed using copies of the extracted objects, as will be discussed later, reconstruction quality is a key factor to be considered during the edge labelling process. As such, external thresholds to guarantee a certain reconstruction quality are incorporated within the learning algorithm. Chapter 4 discusses this algorithm along with the external thresholds in more detail.

The objects are extracted as frequent subgraphs by mining the labelled graph, as shown in figure 3.5. Each subgraph is known as an instance of the object. The details of the mining algorithm itself are discussed in chapter 5. Each object is characterized by the recurring edge labels involved in its instances. Each object instance is characterized

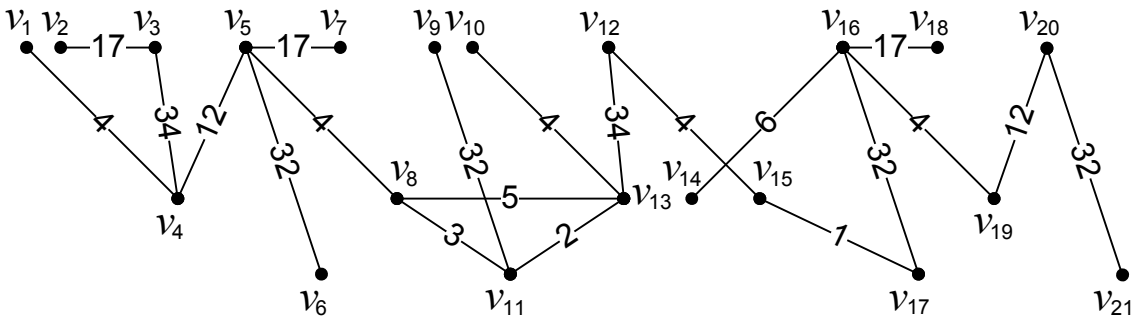


Figure 3.4: Labelled Graph

by the vertices involved in its subgraph.

Every time an object is extracted, it is recorded and the associated vertices are removed from the original graph. The graph mining process then repeats on the remaining graph to extract another object. This procedure is repeated until no more objects can be found. When the algorithm terminates, the spikes associated with the remaining vertices are recorded as residual and kept along with the extracted objects.

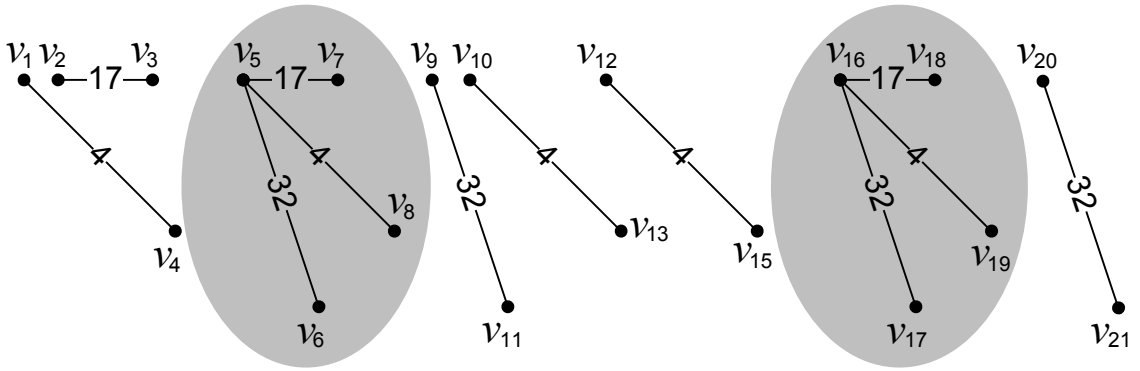
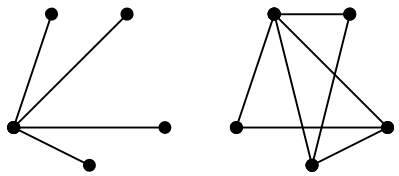


Figure 3.5: Co-occurrence Principle

A restriction is placed on the type of extracted subgraphs by only allowing stars to be valid objects. A star is a subgraph where a central vertex is directly connected to every other vertex. Figure 3.6 shows how a star differs from any other subgraph. This restriction is imposed for two reasons. First, by forcing the instances of an object to be stars, it is guaranteed that the error - when comparing a vertex and central vertex (i.e. an edge) in one instance to the equivalent two vertices in another instance of the object - falls below the external thresholds used by the learning algorithm during the edge labelling



(a) Star (b) Subgraph

Figure 3.6: Star vs. Subgraph

process. The second reason for imposing stars is for object recording purposes. For each object that is extracted, two main components must be recorded: a representative object and the locations of the object instances. By doing so, the spikegram can be re-synthesized by placing copies of the representative object at the appropriate positions. The position - also known as anchor point - of each instance must be the coordinates of a spike corresponding to a selected vertex from the subgraph representing the instance. This raises the question of which vertex to select. By ensuring that each object instance is a star, the anchor point can be selected as the central vertex since all vertices in the instance are directly linked to it. In other words, all other vertices in the instance are directly expressible as a function of the anchor point if the central vertex is chosen as that point. Chapter 6 further details how the objects are recorded after their extraction.

From a conceptual point of view, the proposed object extraction method uses the principle of co-occurrence as its operational backbone. In real life situations, evidence of an event is based on hints coinciding together. This is analogous to the famous expression "Where there's smoke, there's fire". When the same hints coincide - or co-occur - together in more than one place, evidence of a recurring event whose presence is more than just random is established. Two factors determine the strength of an event: the number of hints involved in the event (i.e. the size of the event) and the number of times these hints co-occur together (i.e. the frequency of the event). In graph notation, these hints are represented by the labelled edges. Two edge labels are said to be co-occurring if they have a vertex in common. The co-occurrence principle just described is well illustrated in figure 3.5. Looking at the edges labelled '4', '17', and '32', it can be seen that '4' occurs five times, '17' occurs three times, and '32' occurs four times. However, the frequency of the individual labels does not provide much information. It is only when '4','7', and '32' co-occur together - which only occurs in two places - that evidence of some event becomes evident, although the event itself is not known. Generalizing this example, the enforcement of stars makes the discovery of frequent large objects difficult since the

number of required co-occurring edges grows linearly as the number of vertices involved in each instance increases by one. To discover an object whose instances have k vertices, $k-1$ co-occurring edges are required to form a star.

A key advantage with the proposed scheme is that it makes use of the co-occurrence principle to eliminate the influence of outliers when recording each discovered object. Taking figure 3.5 as an example, rather than using all edges labelled as '4' to record the concerned part of the discovered object in the graph, only the two edges labelled as '4' which are involved in the two instances of the object are used in its recording, as further detailed in chapter 6. This is because, with regards to this particular object, the other edges - even though they have the same label '4' and therefore represent the same type of relationship - are viewed as outliers which incorrectly influence the recorded object since they are not included in any of its instances. As such, the learning algorithm in the proposed scheme can be viewed as a hypothesis placement step when labelling the edges in the graph. The co-occurrence principle then acts as a robustness criterion which confirms or rejects the hypothesized labelling based on whether the labelled edges co-occur to form an object or not, and only involves those which do co-occur when recording the object.

The proposed scheme is comparable to two other methods of particular interest in the audio literature. In [36], the authors make use of a pattern extraction technique known as Frequent Episode Discovery (FED) to find audio objects from a spikegram. The main difference between this approach and the proposed one is that the extracted patterns are sequences as opposed to stars. A sequence, which can be represented graphically as a chain of vertices, is not as robust as a star since not all vertices are directly connected to a central (i.e. reference) vertex. Secondly, in [37], audio objects are extracted by partitioning an audio waveform into contiguous segments. A perceptual measure is then employed to evaluate the similarity amongst the segments. The segments which are deemed similar are viewed as instances of the same audio object. They are replaced in the waveform with copies of a single representative instance at the appropriate locations. In contrast to this method, where segments are positioned contiguously, the use of graph theory combined with the spikegram in the proposed method allows for audio object instances to contain non-adjacent spikes from the spikegram. This in turn allows for instances to be interlaced and thus opens the door for a greater number of possible objects.

Chapter 4

Graph Formation

4.1 Introduction

The creation of a labelled graph involves two steps: the formation of edges to link vertices and the use of a learning algorithm to label these edges. To form edges in the graph, a mechanism must be established to numerically define the properties of each edge. Section 4.2 further discusses this mechanism.

The learning algorithm aims to find the optimal categorization for the edges by considering them as entities and placing them in different groups. The entities in the same group are then assigned the same label. Because the correct group assignment for any of the entities is not known a priori, the above problem is known as an unsupervised learning problem. In contrast to unsupervised learning problems, supervised learning algorithms make use of entities whose group assignment is known a priori to first define the categorization space and then classify unseen entities in this fixed space. Since the categorization space cannot be defined a priori in unsupervised learning, it is updated throughout the learning process.

One of the most commonly used subset of unsupervised learning algorithms is the clustering family. Other popular unsupervised learning algorithms include Neural Networks (NNs), Self Organizing Maps (SOMs), and the Expectation-Maximization (EM) algorithm. Since the learning algorithm presented in this chapter is based on clustering, these other methods are not surveyed in this thesis and the reader is instead referred to [38] for additional information. The NMF and PRG algorithms surveyed in chapter 3 also fall in the category of unsupervised learning algorithms.

The clustering family includes the traditional k-means algorithm [39] which operates

as follows. A predefined number of group centroids is first chosen. Each entity is then checked against all group centroids and assigned to the one with which is matches best. Once all entities are assigned, each group centroid is updated based on the entities which are assigned to it. The entities are then reassigned based on the new group centroids. This process repeats for a desired number of iterations or until none of the entities are reassigned to a different group when going from one iteration to the next. Many variants of this algorithm exist, for example where the starting group centroids are chosen differently or where the centroids are updated after each entity assignment rather than waiting for all entities to be assigned.

The first problem when assigning an entity to a group with most learning algorithms - whether supervised or not - is that they only look at which group offers the best match. This does not say anything about how well the entity in question matches with the assigned group but only indicates that it matches better than with any of the other possible groups. In applications such as the one presented in this thesis, where audio quality must be preserved in contrast to pure audio analysis applications, it is empirical that a lower bound as to how well an entity must match with its assigned group be enforced. The second problem with most learning algorithms is that the number of groups must be defined prior to launching the algorithm. Predefining the number of groups directly impacts the final entity assignments. If not done correctly, it can lead to incoherent groupings as mentioned for NMF in chapter 3. Although methods such as the one in [40] exist to help choose the number of groups, these schemes still rely on a priori information.

To address both concerns in the previous paragraph, quality threshold clustering was introduced in [41]. With this clustering algorithm, all entities which are assigned to a same group are guaranteed not to deviate from each other by more than one or more predefined thresholds. In other words, each entity is assigned to the group with which it matches best such that it meets a certain quality guarantee. This formulation is similar to that of a constrained optimization problem. Because of this threshold requirement, the number of clusters changes throughout the clustering process. This removes the need to specify the number of clusters a priori and allows for a variable number of clusters. Although the thresholds must be known a priori, it is often easier in application-specific scenarios to specify them rather than define the number of clusters. Chapter 8 details the thresholds for a spikegram-specific experiment. Section 4.3 further describes quality threshold clustering and presents some modifications to address its pitfalls when working with multidimensional entities. The resulting algorithm is combined with the

edge formation step, as shown in figure 4.1, to outline the overall graph formation process.

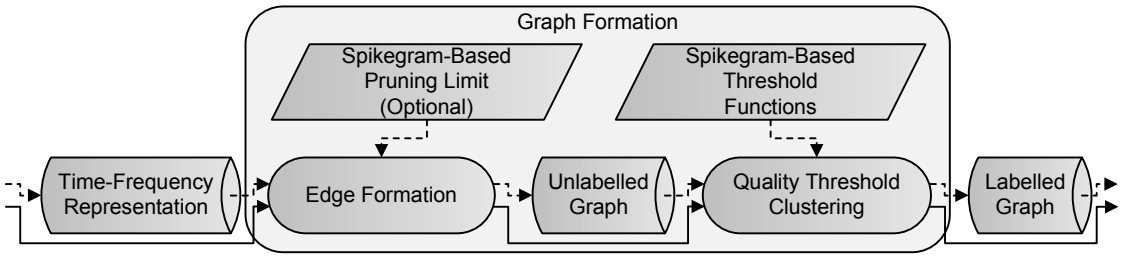


Figure 4.1: Graph Formation Block

4.2 Edge Formation

As mentioned in chapter 3, each edge in the graph is characterized based on the two vertices which it connects. A quantitative description of this characterization is established by associating a feature vector $\mathbf{z}$ to each edge. The number of features in the vector as well as what they represent varies based on the application-specific parameters associated with the vertices. Chapter 8 details the description of features vectors for a spikegram-specific experiment. To avoid loss of generality in this chapter, each feature vector is said to have D features (i.e. D dimensions). Together, all feature vectors form a feature set Z.

In general, the created graph contains edges directly linking any two vertices, thereby forming a complete graph. However, as an optional step, some of the edges can be discarded from the graph prior to the labelling process if one or more features in their feature vectors exceed a predefined limit. This pruning limit is motivated by application-specific a priori knowledge that it is impossible to mine subgraphs comprised of certain types of edges from the graph. The pruning limit for a spikegram-specific experiment is addressed in chapter 8.

4.3 Quality Threshold Clustering

Quality Threshold (QT) clustering operates as shown in figure 4.2. This is an iterative process where each QT iteration gives rise to one cluster as follows. Starting with an

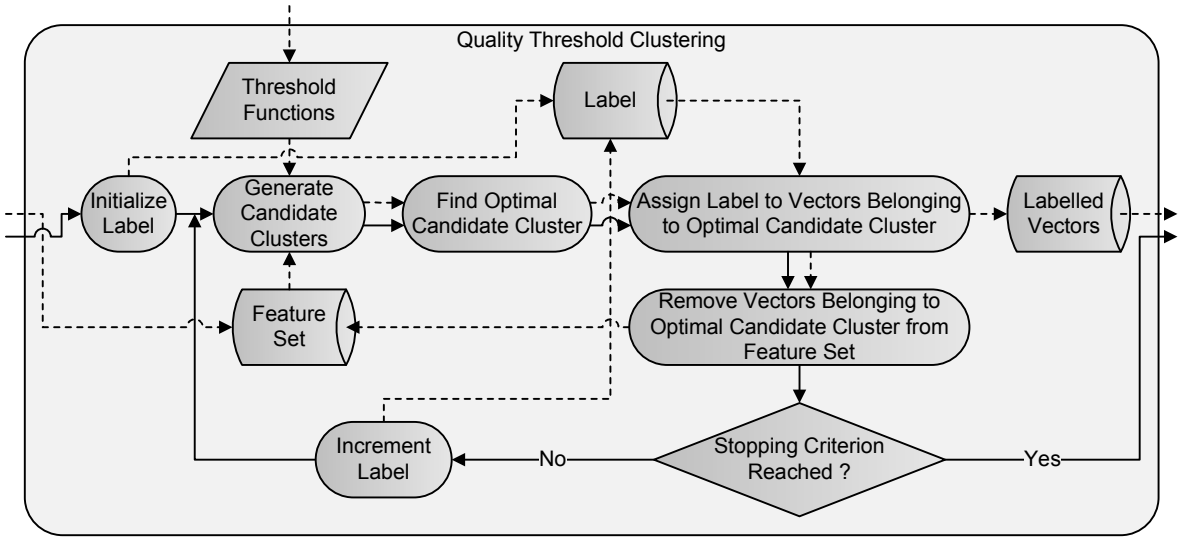


Figure 4.2: Quality Threshold Clustering Block

input feature set, a group of candidate clusters are first generated. Each candidate cluster is iteratively grown, starting with a vector from the feature set, by finding and including the next vector which minimizes the increase in cluster diameter. This growth process terminates when no more vectors can be added to the candidate cluster without having the diameter exceed a predefined threshold T. The diameter of a candidate cluster is defined as the largest discrepancy between any two vectors in the cluster. This discrepancy can be measured with a variety of metrics such as the Euclidean distance (i.e. l2-norm) or the city block distance (i.e. l1-norm), amongst others. Each vector in the feature set serves as the starting point for a candidate cluster. As such, the total number of candidate clusters is equal to the size of the feature set. It should be noted that the same feature set is used to form all candidate clusters. In other words, vectors used in one candidate cluster remain available when forming other candidates. The candidate clusters are thus not mutually exclusive. Mutually exclusive candidate clusters are not desirable since vectors included in a first candidate cannot be included in a second even though they may be better suited for the latter. This would prevent the algorithm from presenting all viable candidate clusters and hence run the risk of missing out on presenting the best solution. By allowing vectors to be included in more than one candidate cluster, the algorithm avoids this problem and guarantees that the best solution exists as one of the candidate clusters. Once all candidate clusters are formed, the optimal candidate is selected as the cluster to be extracted. The optimal cluster is defined as the one with

the largest cardinality (i.e. the largest number of vectors). The vectors associated with the selected cluster are given the same label and then removed from the feature set. The reduced set is then returned as input for the next QT iteration. This process repeats until either a desired number of clusters are extracted, until the cardinality of the latest extracted cluster falls below a predefined value, or until the input set is empty. The extracted clusters are non-overlapping as only one candidate cluster is selected at each iteration and its vectors are not made available to form the candidate clusters in the next iteration. The algorithm guarantees that a cluster extracted at any iteration cannot have a larger cardinality than any of those extracted at previous iterations. This is because the set of candidate clusters presented at any iteration consists of candidates which either remain unchanged or which are subsets of candidates from the previous iteration. In the latter case, the vectors missing from the subsets belong to the cluster extracted in the previous iteration.

The main drawback with QT clustering is its computational complexity due to the many candidate clusters which must be formed at each QT iteration. One of the most cumbersome operations when growing each candidate cluster is the search for the next feature vector which will minimize the increase in diameter. A proposed solution to minimize the search time, when working with a unidimensional feature set, is to sort the features. The advantage of using a sorted set is that the features which are most alike with each other are then positioned adjacently meaning that the contents of a candidate cluster is always contiguous in the sorted set. Thus, when growing a candidate cluster, the next feature which minimizes the increase in diameter is always adjacent to those already in the candidate. This implies that selecting the next feature only requires two checks - one for the feature located immediately above the candidate and one for the feature just below the candidate - as opposed to checking all features in the set which do not already belong to the candidate. Furthermore, the feature set needs to be sorted only once to generate all candidate clusters. Figure 4.3 illustrates the growth of a candidate cluster when using a unidimensional sorted set. In this example, the metric used to measure discrepancy is just the absolute difference between the values of two features.

When working with a multidimensional feature set, it is often desired to examine the discrepancy between two feature vectors on a per dimension basis rather than using metrics which relate the discrepancy as a single value by combining the discrepancies along each dimension of the vectors (i.e. the Euclidean or city block distances, amongst others). This is because these metrics fail to indicate how the overall reported discrepancy is distributed along each dimension. As such, two pairs of vectors whose overall

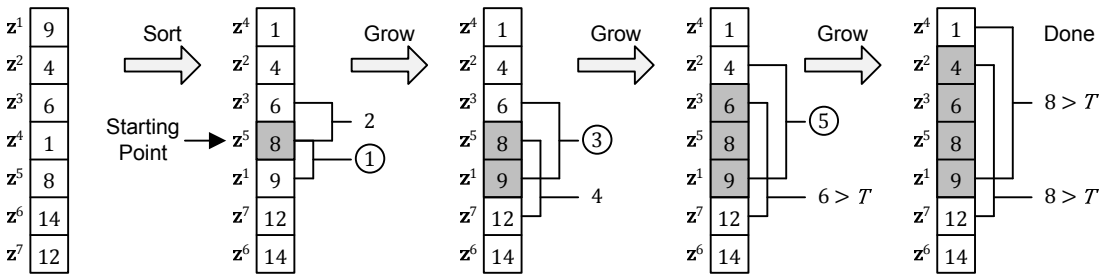


Figure 4.3: Unidimensional Candidate Generation with $T = 5$

respective discrepancies are the same may have very different discrepancies along each dimension. When examining discrepancies on a per dimension basis, a candidate cluster has diameters along each dimension which must not exceed respective thresholds. For such situations, section 4.3.1 presents an extension of the sorting scheme outlined in the previous paragraph. The extended scheme is based on the sequential sorting of the feature set along each dimension in a hierarchical fashion.

4.3.1 Hierarchical Candidate Cluster Generation Scheme

The main problem when dealing with a multidimensional feature set is that the vectors cannot be sorted according to all dimensions at once. As such, the proposed scheme begins by sorting the feature set according to the first dimension. Each vector in the sorted set serves as the starting point for a candidate cluster. Each candidate is iteratively grown according to the first dimensional component of the feature vectors. In other words, the next feature vector whose first dimensional component minimizes the increase in diameter for the first dimension is found and included in the candidate cluster. Because the vectors are sorted according to this first dimension, this next vector is always adjacent to the those already in the candidate, thereby minimizing the search time. The growth process terminates when no more vectors can be included within the candidate cluster without having the first dimension diameter exceed its respective threshold.

The vectors included in a candidate cluster based on their first dimensional components are then re-sorted according to their second dimensional components. The candidate cluster is then used to form new candidates, where the second dimensional component of each vector from the incoming candidate cluster serves as the starting point for each new candidate. Since it is already known that the incoming candidate cluster has a first dimension diameter which does not exceed its respective threshold,

the task is now to determine which vectors from this candidate have second dimensional components that, when grouped together, yield a second dimension diameter which does not exceed its respective threshold. Since the new candidates are based on subsets of the incoming candidate cluster, it is guaranteed that these new candidates have a first dimension diameter which is less than or equal to the original diameter of the incoming candidate cluster and therefore less than or equal to its respective threshold. Each new candidate cluster is iteratively grown according to the second dimensional component of the feature vectors from the incoming candidate cluster. Here again, having the vectors sorted according to the second dimension minimizes the search time. The growth process terminates when no more vectors from the incoming candidate cluster can be included within the new candidate without having the second dimension diameter exceed its respective threshold.

The above process, described for the first two dimensional components of a feature set, repeats for each component until the last dimension of the feature set is reached. At each dimension, an incoming candidate cluster forms new candidate clusters which are subsets of the latter based on the threshold of the current dimensional component. The set of candidate clusters is only valid once all dimensional components are traversed because only then do the diameters of each candidate cluster satisfy all respective dimensional thresholds. Once a valid set of candidate clusters is established, the optimal candidate is selected, extracted, and the reduced feature set is returned as input for the next QT iteration, as per the original QT algorithm. Figure 4.4 shows an example of the proposed hierarchical candidate generation scheme in a multidimensional context. This example demonstrates a critical flaw of the iterative growth process for the candidate clusters in that it does not produce a set of candidates which spans the entire solution space when working with a multidimensional feature set, even though the candidates in the set overlap. It can be seen in figure 4.4 that the four new candidates formed from the second dimensional component analysis of the feature vectors are smaller than the incoming candidate cluster obtained from the first dimensional component analysis. This often happens in practice as well. The issue with this phenomenon is that the second dimension threshold which restricts the second dimension diameters - and hence the cardinality - of the new candidates often also decreases the first dimension diameters simply because there are less vectors in the new candidates, thereby increasing the likelihood that the first dimension diameters of the new candidates are less than that of the incoming cluster. This can be thought of as a feedback effect from the second dimensional components towards those of the first dimension. These reduced first dimension diame-

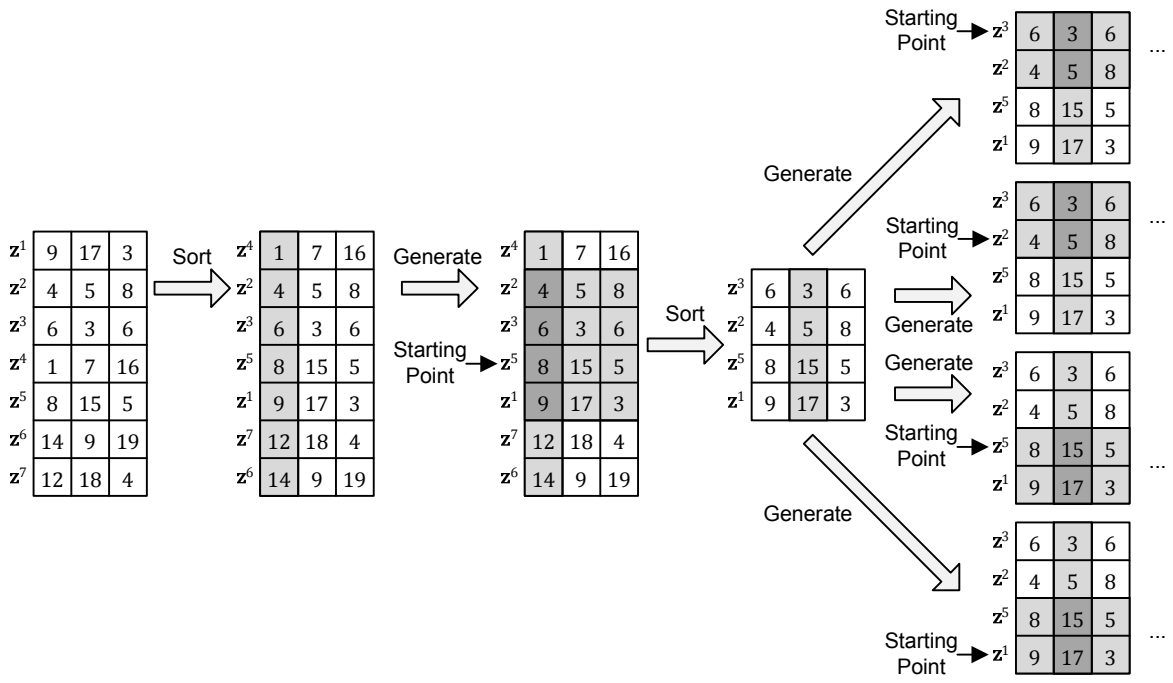


Figure 4.4: Multidimensional Candidate Generation with $T_1 = 5$ and $T_2 = 3$

ters introduce the possibility of other vectors from the original feature set being included in the new candidate clusters while still having the diameters of these candidates respect both the first and second dimension thresholds. Going back to the example in figure 4.4, the new candidates involving vectors $\mathbf{z}^1$ and $\mathbf{z}^5$ could now also include vector $\mathbf{z}^7$ and have diameters which still respect both thresholds. However, the hierarchical generation scheme states that new clusters can only be subsets of an incoming candidate cluster and it is already known that all vectors from this cluster capable of producing new candidates which respect the second dimension threshold have already been considered. Hence, the incoming candidate cluster bottlenecks the new candidates by preventing them from including vectors from the original feature set which now could be done due to their reduced first dimension diameters. This situation does not pose a problem as long as the possible solution paths stemming from these inclusions are investigated as part of other candidate clusters generated from the other first dimensional starting points in the original feature set. Unfortunately, due to the nature of the iterative growth process, this does not happen. Figure 4.5 shows the candidate clusters generated from all starting points - after duplicates and subsets contained in other candidates are removed - using the iterative growth process for the first dimension components of the vectors in the

example of figure 4.4. Relating back to this example, it can be seen that a candidate

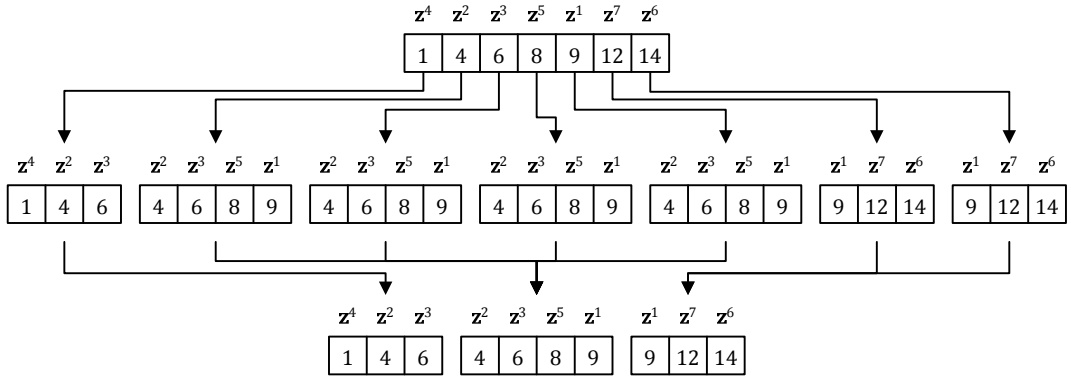


Figure 4.5: Candidate Clusters from First Dimensional Component in Original Scheme

involving the first dimensional components of vectors $\mathbf{z}^1$, $\mathbf{z}^5$, and $\mathbf{z}^7$ together is not generated. The above mentioned scenario does not only occur between the first and second dimensional components but generally between any two dimensional components.

In the original iterative growth approach, a candidate is grown by finding the next feature which minimizes the increase in diameter by performing a bidirectional search and looking at features whose values are both greater and smaller than that of the feature used as a starting point. The issue with this approach is that the bidirectional nature of the search entertains the possibility that candidate clusters originating from different starting points may end up with the same contents by growing towards each other's starting points. As an example, consider two starting points where the value of the first is less than that of the second. With a bidirectional search, a situation could occur where the candidate cluster stemming from the first starting point would only grow upwards, by including features whose values are greater than that of the starting point, whereas the candidate stemming from the second starting point would only grow downwards by including features with values less than that of the starting point. Hence, both candidate would end up with same feature vectors. This issue is not a problem in a unidimensional context since the original iterative growth approach guarantees that each candidate cluster contains the largest number of features without exceeding the threshold. This means the candidate clusters which are not presented have a smaller cardinality and hence will never be selected as the cluster to be extracted. As such, even though the set of candidate clusters produced in this context may not span the entire solution space, the best solution is still guaranteed to exist within the set. However, the same cannot be said

in a multidimensional context due to the feedback effects mentioned earlier. To address this problem, a new iterative growth approach is proposed. In the proposed approach, the feature used as the starting point is set as the lower bound for the candidate diameter and other features are included in the candidate by performing a unidirectional search and looking only at features whose values are greater than that of the feature used as a starting point. Figure 4.6 shows how this process works in a unidimensional case. As

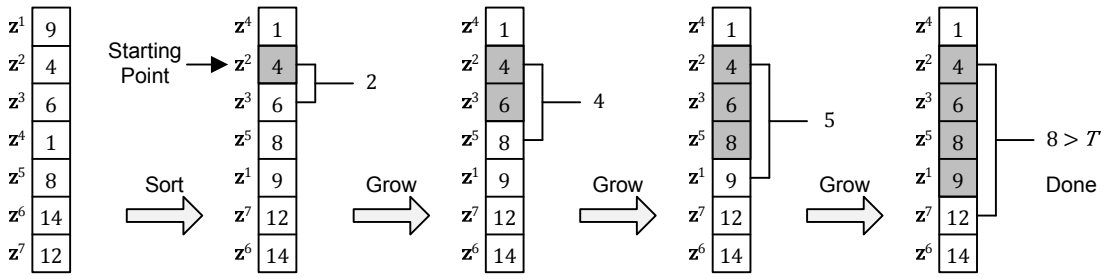


Figure 4.6: New Unidimensional Candidate Generation with $T = 5$

a side benefit, the unidirectional nature of the new scheme improves the computational complexity in two ways: when choosing the next feature to include in the candidate cluster, only one diameter calculation needs to be performed as opposed to two in the original scheme and the diameter only needs to be checked against the threshold rather than both the threshold and the other computed diameter. It should be noted that in the new scheme, the next added feature may not always be the best choice to minimize the increase in diameter. All that is known is that the next added feature yields a diameter which is less than or equal to the threshold. However, the issue of the original scheme is now solved since different starting points can never yield candidate clusters with the same content. Hence, the set of candidate clusters spans the entire solution space. In a multidimensional context, this means that the feedback phenomenon discussed earlier no longer poses a problem since the possible solution paths stemming from the inclusions bottlenecked by an incoming cluster are now guaranteed to be investigated as part of other candidate clusters generated from the other first dimensional starting points in the original feature set. Figure 4.7 shows the candidate clusters generated from all starting points - after duplicates and subsets contained in other candidates are removed - using the new growth process for the first dimension components of the vectors in the example of figure 4.4. Relating back to this example, it can be seen that with the new scheme, a candidate involving the first dimensional components of vectors $\mathbf{z}^1$, $\mathbf{z}^5$, and $\mathbf{z}^7$

together is generated. Since the new unidirectional growth process guarantees that all

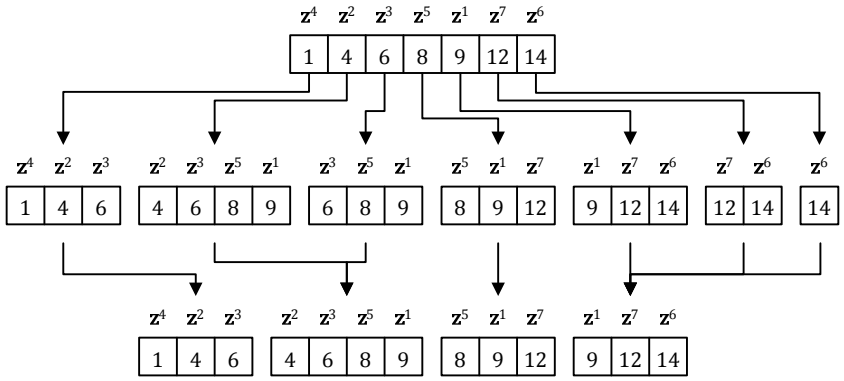


Figure 4.7: Candidate Clusters from First Dimensional Component in New Scheme

possible search paths are investigated in a multidimensional context, the order in which the dimensional components are handled also becomes irrelevant. Figure 4.8 shows how the new iterative growth process operates when combined with the proposed hierarchical candidate generation scheme. A second advantage of investigating all possible search

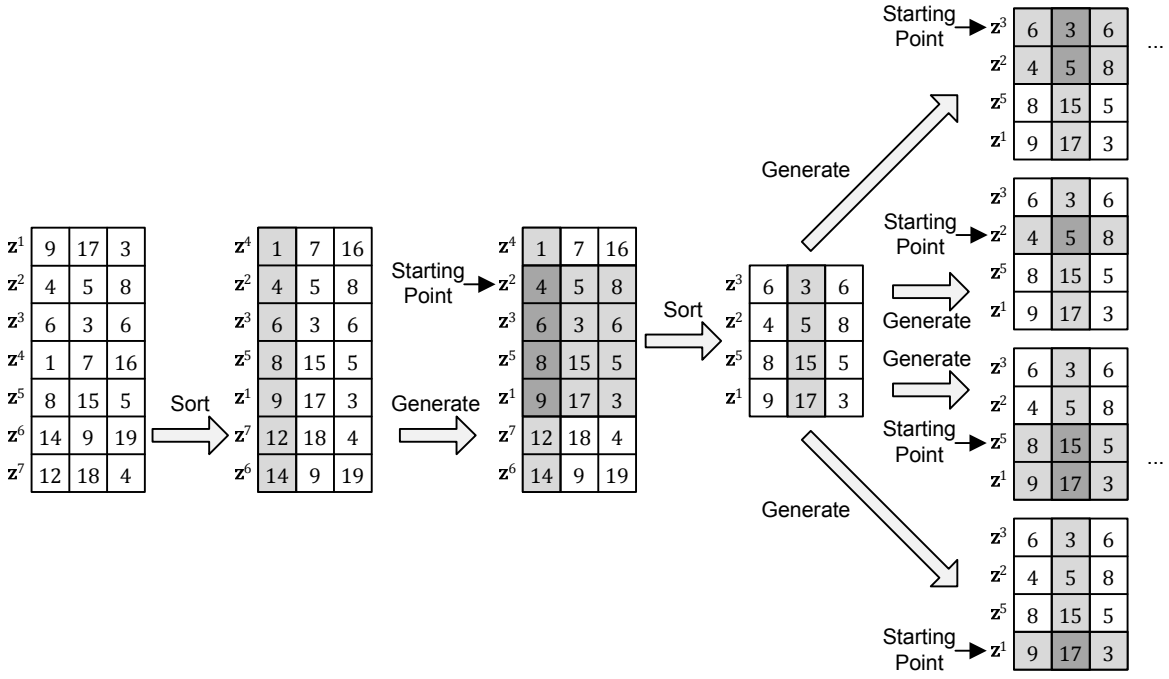


Figure 4.8: New Multidimensional Candidate Generation with $T_1 = 5$ and $T_2 = 3$

paths with the new unidirectional growth process is that a new set of candidate clusters does not need to be generated from scratch using the hierarchical candidate cluster generation scheme at each QT iteration. Instead the new set of candidate clusters at the next QT iteration is simply obtained by taking the current set of candidate clusters and removing the feature vectors involved in the extracted cluster from all the candidate clusters. If the deletion of one or more feature vectors in a candidate cluster causes a dimensional component of the candidate to now be capable of accepting new vectors from the original feature set before exceeding its respective threshold and all the while keeping other dimensional thresholds in check, the possible solution paths stemming from these inclusions are guaranteed to be investigated as part of other candidate clusters. This is a big benefit from a computational complexity standpoint. Lastly, it should be noted that although the multidimensional candidate generation scheme is hierarchical, the overall QT algorithm still belongs to the family of partitional clustering algorithms because of the manner in which the clusters are extracted (i.e. each extracted cluster is not formed by splitting or grouping previously extracted clusters but rather by choosing an optimal solution from a new set of candidates).

4.3.2 Optimal Candidate Cluster Decision Criterion

A problem with the original QT algorithm is that a tie breaking procedure is not outlined to indicate which cluster should be selected for extraction when more than one candidate cluster is tied for the largest cardinality. In practice, this situation can occur quite often. As a solution to this problem, the candidate cluster with the smallest diameter among those tied for the largest cardinality is selected to break the tie. This is straightforward when the feature set is unidimensional. In a multidimensional context, the following procedure is employed for each candidate cluster which is tied for the largest cardinality. As the diameters of the candidate cluster may not be in the same dynamic range, these are first normalized by dividing each diameter by its respective threshold. It should be noted that during this normalization process, weights may be optionally added to emphasize certain dimensions. In this normalized setting, a diameter of one indicates that the diameter of the candidate cluster taken along the dimension in question is equal to the respective threshold. The candidate cluster whose largest normalized diameter is minimal among those tied for the largest cardinality is selected to break the tie. By doing so, it is guaranteed that the other diameters of the selected cluster are also smaller than the largest normalized diameters of the other candidate clusters.

Chapter 5

Subgraph Extraction

5.1 Introduction

The mining of frequent subgraphs is carried out, as shown in figure 5.1, using a star extraction algorithm with the help of an application-specific cost function C. Chapter

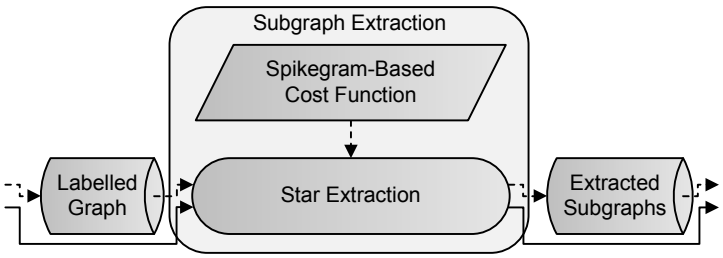


Figure 5.1: Subgraph Extraction Block

8 details the cost function used in the star extraction process for a spikegram-specific experiment.

The labelled graph input to the star extraction algorithm is stored as a square adjacency matrix. The rows and columns of this matrix represent the indices of the vertices in the graph. As a convention, the row indices are used to represent the left vertex of any edge while the columns are used for the right vertex. For two vertices v_i and v_j in the graph, where v_i is located to the left of v_j, the edge linking them is indexed as $e_{(i,j)}$ and its label is defined as $l\left(e_{(i,j)}\right)$. This label is placed in the matrix entry corresponding to row i and column j. Due to the convention established in chapter 3, where the vertices are indexed in increasing order from left to right in the graph, the right vertex of any

edge always has a greater index than the left vertex. As such, only the upper triangular section of the matrix is used. The lower triangular section of the matrix along with the main diagonal are filled with zeros. Based on this adjacency matrix, the star extraction algorithm operates as shown in figure 5.2. The algorithm is iterative in which one fre-

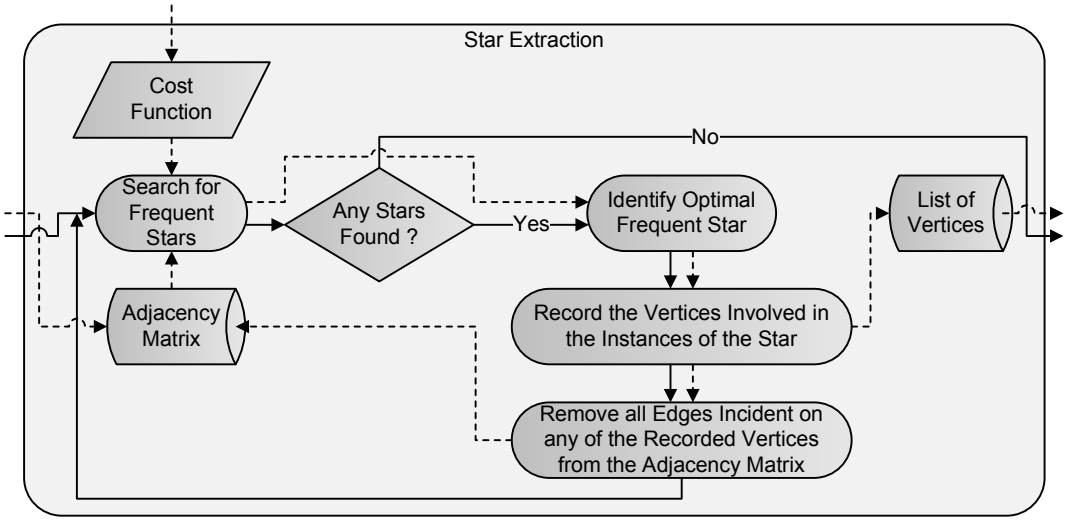


Figure 5.2: Star Extraction Block

quent star is extracted at each iteration. A star search, as detailed in section 5.2, first finds all frequent stars in the graph and computes an extraction cost for each of them. If frequent stars are found, the one with minimal extraction cost is then recognized and its instances are recorded and removed from the adjacency matrix as explained in section 5.3. Lastly, the new adjacency matrix, which now represents the remaining graph, is returned as input for the next iteration. The process repeats until no more frequent stars are found.

5.2 Frequent Star Search

The topic of frequent subgraph discovery is well documented. Two of the most common algorithms in this field are FSG and GSPAN, developed in [42] and [43] respectively. Both these algorithms find frequent subgraphs in a database of graphs. This is known as subgraph discovery in a transactional setting where each graph is regarded as a transaction. As such, these algorithms cannot be used to tackle the problem presented in this chapter where frequent subgraphs are mined from a single graph. This is because the

method used to count the frequency of subgraphs in a transactional setting differs from the one used in a single graph setting.

In [44], two algorithms known as HSIGRAM and VSIGRAM were introduced to mine frequent subgraphs in a single graph setting. However, these algorithms, along with FSG and GSPAN, are designed to mine all frequent subgraphs as opposed to focusing only on frequent stars. Since stars are a very specific type of subgraph, focusing solely on their discovery allows for many simplifications to these algorithms which significantly reduce the search space and time.

The first algorithm to mine a specific type of subgraph, known as CLAN, was presented in [45]. CLAN is designed to mine frequent cliques. A clique is a subgraph where every vertex is directly connected to every other vertex. Since CLAN can only find frequent cliques in graphs with labelled vertices, a version - dubbed MCM - which can handle graphs with labelled edges was later proposed in [46]. However, much like FSG and GSPAN, both these algorithms operate in a transactional setting and require modifications to handle a single graph as input. Furthermore, mining frequent stars as opposed to frequent cliques allows for even further simplifications to the MCM algorithm. To address both these issues, an algorithm for frequent star discovery in a single graph setting is proposed in this section. This new procedure is based on the MCM algorithm and draws from the frequency counting method for a single graph context used in the HSIGRAM and VSIGRAM algorithms.

In the proposed algorithm, a frequent star with k vertices is referred to as a size-k frequent star. The algorithm uses a recursive depth first approach to find all frequent stars. As a starting point, each labelled edge in the graph is marked as a candidate instance. Based on the frequency of the candidate instance labels (i.e. the edges labels), the frequency counting method outlined in section 5.2.2 recognizes all size-2 frequent stars (i.e. frequent single edges) in the graph. The entries in the adjacency matrix which do not correspond to edges that are instances of any size-2 frequent stars are set to zero. The algorithm then recurses for each size-k frequent star. Each instance of the size-k frequent star gives rise to many size-$k+1$ candidate instances, as shown in the example of figure 5.3 where the instances of a size-2 frequent star each generate two and four size-3 candidate instances respectively. Each candidate instance is described by an instance identifier and the indices of the vertices associated with the instance, as further detailed in section 5.2.1. All size-$k+1$ candidate instances stemming from the instances of the same size-k frequent star are then considered together, thus forming a list of instance identifiers and respective associated vertices. The candidate instances which have the

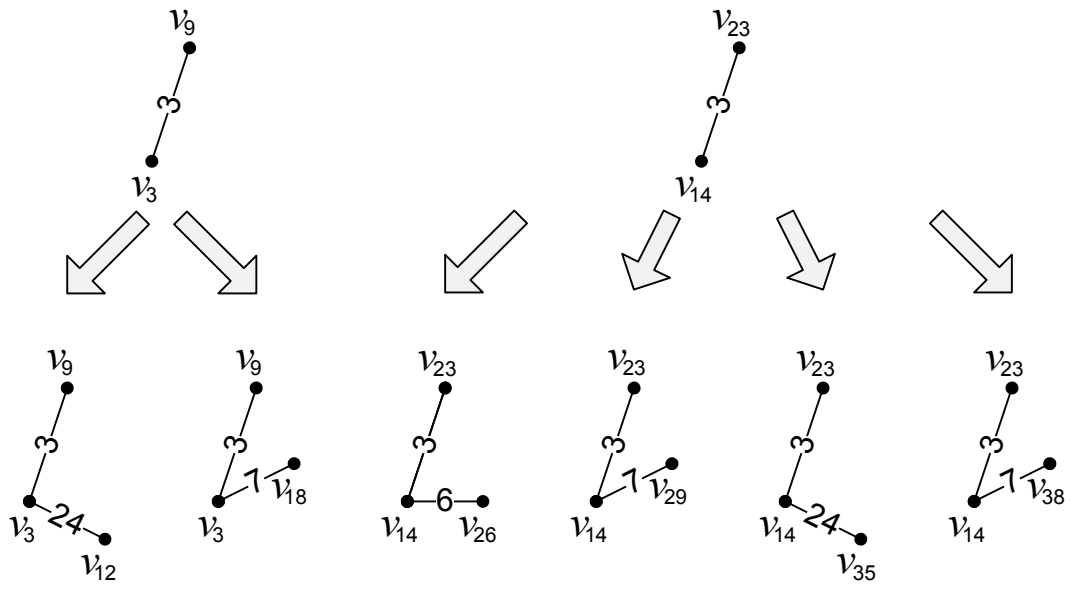


Figure 5.3: Generation of Candidate Instances

same identifier that is deemed frequent, as per the frequency counting method of section 5.2.2, are recognized as the instances of a new size-$k+1$ frequent star, as shown in figure 5.4 for the example of figure 5.3 where two new frequent stars are recognized, and the above process repeats for this new frequent star. As an optional step, an upper limit k_{max} can be placed on the size of a frequent star in order to restrain the computational time of the algorithm. Every frequent star found throughout the recursive search is assigned an extraction cost as detailed in section 5.2.3.

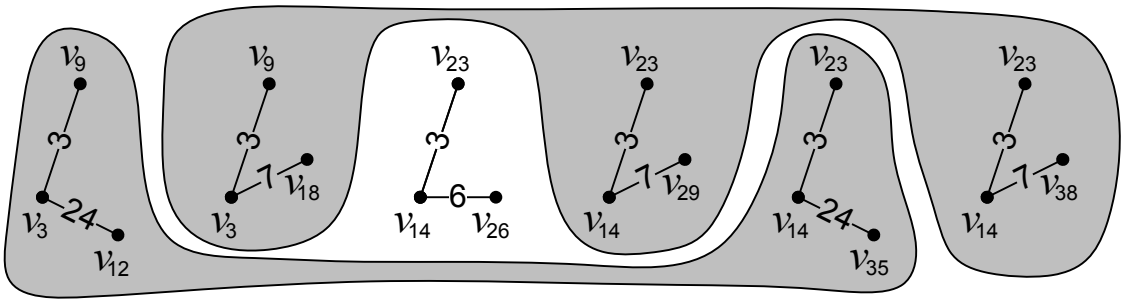


Figure 5.4: New Frequent Star Formation

A key factor in the above algorithm is the usage of the monotonic decreasing observation for frequent stars. Logically, the number of instances of a size-$k+1$ frequent star

cannot exceed the number of instances of a size-k frequent star which resides within it. Conversely, this means that no subsets of the input graph can reside within instances of a size-$k+1$ frequent star unless they have already been recognized as instances of a size-k frequent star. As such, as carried out in the algorithm, only the instances of size-k frequent stars need to be verified when generating size-$k+1$ candidate instances. A similar observation acts as the reasoning behind the deletion of single edges which are not instances of frequent size-2 stars at the beginning of the algorithm: any edge which extends an instance of a size-k frequent star must itself be deemed frequent (i.e. be part of a size-2 frequent star).

5.2.1 Candidate Generation

A method to generate and represent all possible size-$k+1$ candidate instances stemming from a given input size-k instance is detailed in this section. Each candidate instance is a star obtained by extending the input size-k instance by a single vertex.

A key problem when generating candidate instances is that many size-k instances can generate the same size-$k+1$ candidate instance. This is because any size-$k+1$ candidate instance contains many size-k stars within it and each one can be expanded into the former via a different single vertex expansion, as shown in the example of figure 5.5. This leads to multiple copies of a same candidate instance which unnecessarily

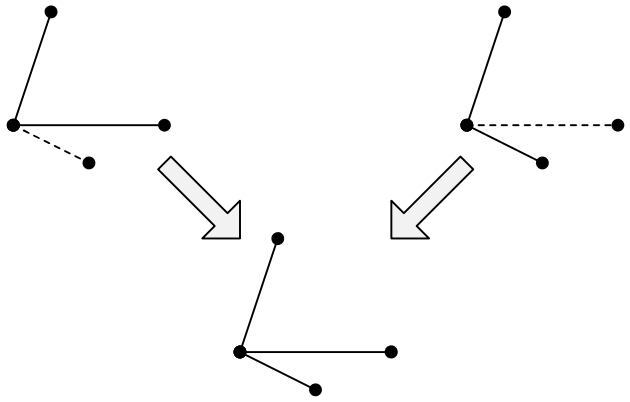
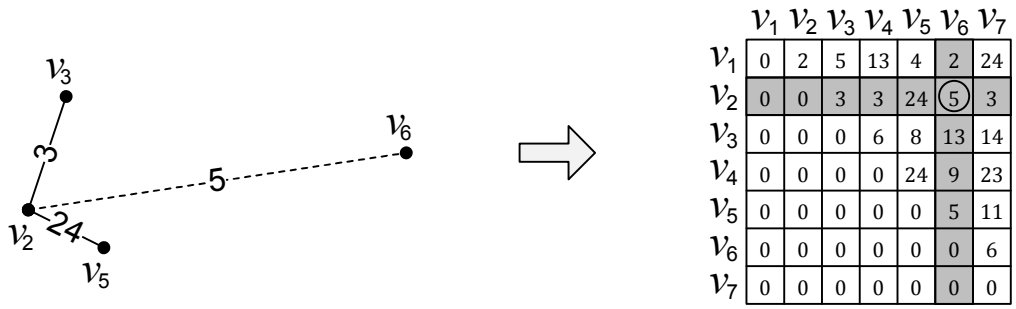


Figure 5.5: Common Star Extension

decreases the performance of the algorithm yet would yield the same solution seeing as the extra copies would be removed during the frequency counting method of section 5.2.2. To address this problem, a Rightward Expansion Rule (RER) is introduced. This rule

states that a vertex extending a size-k instance to yield a size-$k+1$ candidate instance must have an index greater than any of the vertices involved in the size-k instance. Since the vertices in the graph are indexed in increasing order from left to right as a convention, this means that the extending vertex must be located to the right of the vertices involved in the size-k instance. By enforcing this rule, only one size-k instance is capable of generating a particular size-$k+1$ candidate instance unlike the case of figure 5.5. Conversely, any size-$k+1$ candidate instance generated by an input size-k instance is guaranteed not to be generated by any other size-k instance, noting that no two size-k instances are themselves the same. It should also be noted that the employment of the RER automatically implies that the leftmost vertex of a size-k star is its central vertex since it is the left vertex of the first edge used in the formation of the star and any added vertices, which can only be located to its right, must be connected to it in order to maintain a star structure.

The extension of a size-k instance into a size-$k+1$ candidate instance requires the presence of one edge to link the existing central vertex to the new vertex responsible for the expansion. Due to the RER, the left vertex of this edge must bear the index of the existing central vertex while the right vertex of the edge must be the index of the new vertex responsible for the expansion. Based on the above observations and using the adjacency matrix which represents the graph, all size-$k+1$ candidate instances stemming from a size-k instance are found as follows. The row (i.e. the dimension representing the left vertex of any edge) corresponding to the index of the existing central vertex is first located. Each column (i.e. the dimension representing the right vertex of any edge) whose index is greater than the largest index of the k existing vertices (due to the RER) is then searched. When searching a column, only the located row is investigated to see if an edge label (i.e. a non-zero value) exists at this entry. If this is the case, then a size-$k+1$ candidate instance is generated where the index of the new vertex responsible for the expansion is the index of the column. Conversely, if the entry contains a zero value, the algorithm immediately stops the search and proceeds to the next column. As such, each column index has the potential of yielding at most one size-$k+1$ candidate instance. Figure 5.6 shows an example of the above described search process when working with a graph which has seven vertices. In this example, the extension of a size-3 instance into a size-4 candidate instance is investigated by considering the inclusion of the v_6 vertex. As per the outlined process, the v_6 vertex is greater than the largest index of the three existing vertices (i.e. the v_5 vertex). As such, the same process would be repeated, using the same size-3 instance, for the v_7 vertex since it is also greater than the v_5 vertex.



	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_1	0	2	5	13	4	2	24
v_2	0	0	3	3	24	5	3
v_3	0	0	0	6	8	13	14
v_4	0	0	0	0	24	9	23
v_5	0	0	0	0	0	5	11
v_6	0	0	0	0	0	0	6
v_7	0	0	0	0	0	0	0

Figure 5.6: Candidate Instance Search

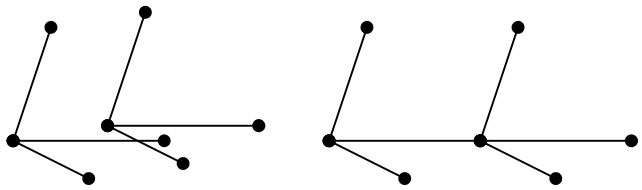
Each candidate instance that is generated is described by two pieces of information. The first piece is an identifier formed with the labels of the edges involved in the size-$k+1$ candidate instance. Because the proposed frequent star search algorithm considers all size-$k+1$ candidate instances stemming from the instances of the same size-k frequent star together in a common list, as mentioned in section 5.2, all candidate instances in the list have the same edge labels except for the single edge which links the existing central vertex of a size-k instance to the new vertex responsible for its expansion. The identifier representing each size-$k+1$ candidate instance is therefore only comprised of a single label corresponding to this edge. Using only a single edge to form the identifier as opposed to a sequence of all k edges belonging to a size-$k+1$ candidate instance not only eliminates the storage of redundant information in memory but also minimizes the required storage space since the identifier composed of a single label remains fixed in size regardless of the number of vertices in the candidate instance in contrast to a k label sequence which grows linearly.

The identifier of each candidate instance says nothing about its positioning in the graph. As such, the second piece of information used to describe a candidate instance is its vertex indices. Here, the indices of all vertices involved in the candidate instance (i.e. including the vertices of the stemming size-k instance) are required. This is because the size-$k+1$ candidate instances, considered together in a common list as per the proposed frequent star search algorithm, may stem from different size-k instances even though the latter all belong to the same size-k frequent star. As a convention, the indices of the vertices for the candidate instance are ordered in an increasing order.

5.2.2 Frequency Counting

Given a list of candidate instances, where each instance is represented by an identifier and the indices of the vertices associated with the instance, the frequency counting method presented in this section aims to find which instances can be collected into groups, where each group contains the instances of a new frequent star. The proposed method examines the given list to find one frequent star at a time. The identifiers and vertices belonging to the examined instances are then removed and the method re-iterates on the reduced list to find another frequent star. The process ends when the given list is empty.

A frequent star must contain a minimum number of instances which do not overlap in the graph. This number is from here onwards referred to as the minimum frequency r_{min}. Two instances are said to be overlapping if they have at least one vertex in common. Figure 5.7 shows how two overlapping instances differ from two interlaced instances, the latter of which is permitted.



(a) Interlaced (b) Overlapped

Figure 5.7: Interlaced Instances vs. Overlapped Instances

To minimize the computational load of the algorithm, the proposed counting method involves two steps: counting the number of instances which have the same identifier and then counting the number of non-overlapping instances for this restricted set. The instances belonging to the restricted set are only submitted to the second step if their frequency is greater than the minimum frequency upon completion of the first step. Otherwise, the identifiers and vertices belonging to the instances in question are removed from the list and the frequency counting method re-iterates on the reduced list to find another frequent star. The above mentioned condition is placed on the execution of the second step because this step is much more time consuming than the first. Logically, the number of non-overlapping instances in a restricted set is always less than or equal to the overall number of instances in this same set. As such, if it is already known, from the execution of the first step, that the overall number of instances is less than the minimum frequency, it makes no sense to count the number of non-overlapping instances in a time

consuming second step as this number is guaranteed to be smaller than the minimum frequency.

The first step of the counting method is straightforward: the first identifier in the list is recorded as a reference and checked against every other identifier in the list to see which ones are exact matches. The instances corresponding to these identifiers form the restricted set for the next step. It should be noted that to further speed up this step, the list containing the identifiers is first sorted. By doing so, the identical identifiers are regrouped in the list thereby restricting the search space.

The second step of the counting method requires the creation of an abstract graph. Each instance in the restricted set is represented by a vertex in this abstract graph. If two instances do not overlap in the original graph, an edge is placed between their two representative vertices in the abstract graph, as shown in the example of figure 5.8. The maximum clique, which is defined as the largest clique in a graph, is then found in

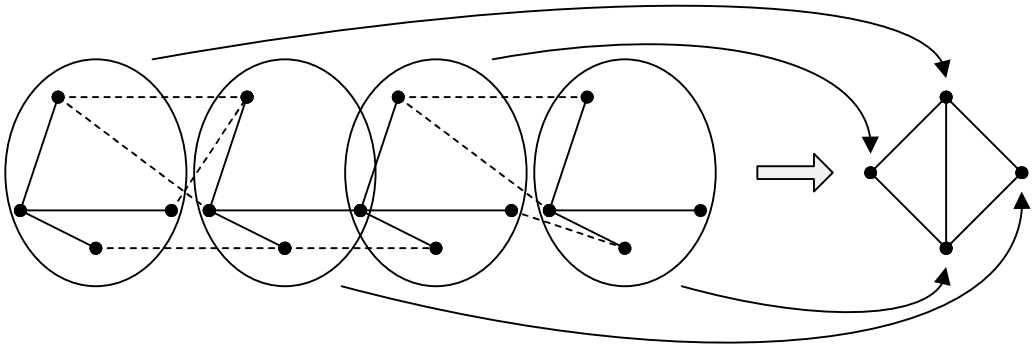


Figure 5.8: Overlap Conversion

the abstract graph. With regards to the original graph, a clique in the abstract graph means that the instances corresponding to the vertices in the clique do not overlap with each other since an edge exists between any two corresponding vertices in the abstract graph. By finding the maximum clique in the abstract graph, the largest number r of non-overlapping instances are found in the restricted set.

The maximum clique problem is well studied in [47], which presents a method for its exact discovery. However, the exact discovery of the maximum clique has been shown to be NP-complete. Its execution time increases drastically for large graphs making it intractable. The maximum clique problem can be reformulated into the Maximum Independent Set (MIS) problem by working with the complement of the abstract graph. The complement graph is obtained from the abstract graph by placing an edge where

no edge exists between two vertices and removing all edges which previously existed. The presence of an edge between two vertices in this complement graph now means that the two instances associated with the vertices do overlap. The MIS in this complement graph is then defined as the largest number of vertices which are not connected via edges. Finding the exact MIS in the complement graph is of the same complexity as the maximum clique problem. However, the authors in [44] use a greedy method to find an approximation to the MIS. This is also the approach taken in this implementation: the complement of the abstract graph is first obtained and the approximate MIS is obtained in the complement graph via this greedy method. For more information about the greedy method, the reader is referred to [44]. The approximate MIS obtained via the greedy method is either of the same size or a subset of the exact MIS. This means that the number of instances found to be non-overlapping is either equal to or less than the exact number of non-overlapping instances. Nonetheless, the approximate MIS obtained via this greedy method is known to give results close to those of the exact MIS [44].

If the size (i.e. number of vertices) of the approximate MIS is greater than or equal to the minimum frequency, the instances corresponding to the vertices involved in the approximate MIS become the instances of a new frequent star. Otherwise, no new frequent star is formed. Either way, the identifiers and vertices corresponding to all the instances in the restricted set - whether involved in the approximate MIS or not - are removed from the list, and the frequency counting method re-iterates on the reduced list to find another frequent star.

5.2.3 Star Cost

Every time a frequent star Q is discovered in the graph, an associated extraction cost is computed. This cost is always based on the frequent star itself and the remaining graph if the instances of the former are removed from it. Further details are application-specific as mentioned earlier in this chapter.

For computational efficiency, only the information regarding the frequent star with minimal extraction cost up to the current point in the search is tracked. That is, unless a size-k frequent star remains to be investigated for size-$k+1$ candidate instances or unless it yields the minimal extraction cost, the frequent star in question is deleted from memory.

5.3 Optimal Star

Once the optimal frequent star is selected, section 5.3.1 details how it should be recorded while section 5.3.2 addresses its removal from the graph.

5.3.1 Recording

The recording of a frequent star requires vertex coordinates rather than edge details. This is because it is the vertices in the graph which are directly mapped to elements in the application-specific problem. As such, to get the required information from the elements associated with the affected vertices in the graph, the vertex indices are the only pieces of information which are recorded. This is performed by recording the frequent star as an object O. The object is a set of instances where $\mathbf{o}^i$ represents the ith instance and o^i_j corresponds to the vertex index of the jth constituent of the ith instance, as shown in figure 5.9 for one of the frequent stars recognized in the example of figure 5.4.

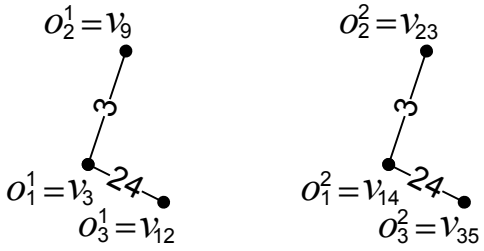


Figure 5.9: Recording of Frequent Star

5.3.2 Removal

When removing a frequent star from the graph, the vertices involved in its instances are first located, as shown by the hollow vertices in the example of figure 5.10. All edges which have at least one end point incident on these vertices are then deleted, as shown by the dotted edges in figure 5.10.

In the adjacency matrix, the above operation is executed by setting the entire row and column associated with a located vertex in an instance to zero, as shown in the seven vertex graph example of figure 5.11 if v_3 is to be a located vertex. This is performed for each of the located vertices.

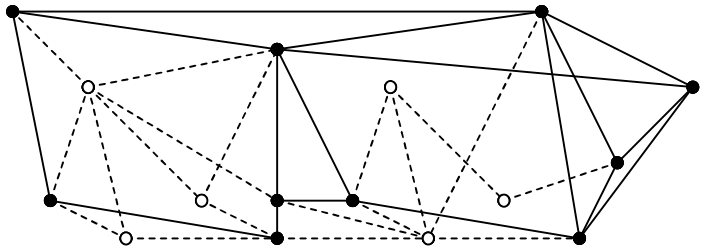


Figure 5.10: Frequent Star Removal

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_1	0	2	0	13	4	2	24
v_2	0	0	0	3	24	7	3
v_3	0	0	0	0	0	0	0
v_4	0	0	0	0	24	9	23
v_5	0	0	0	0	0	5	11
v_6	0	0	0	0	0	0	6
v_7	0	0	0	0	0	0	0

Figure 5.11: Adjacency Matrix Deletion

Chapter 6

Audio Object Recording

6.1 Introduction

The objects extracted in chapter 5 will next be recorded based on their corresponding spikes from the spikegram. Figure 6.1 shows how this is carried out. Given a list of

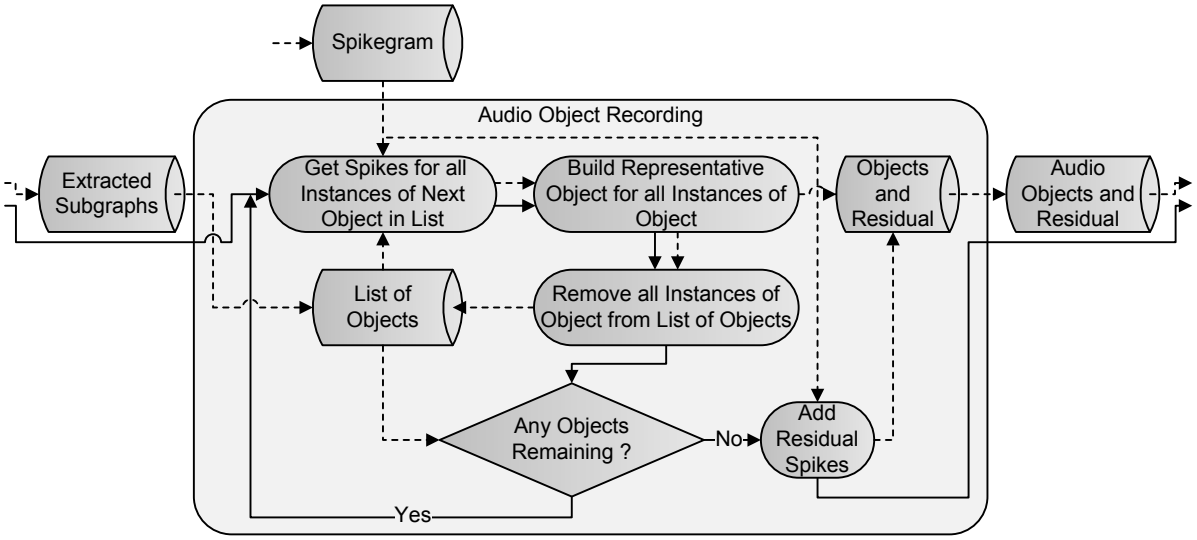


Figure 6.1: Audio Object Recording Block

extracted objects, the spikes corresponding to the instances of the first object in the list are retrieved from the spikegram. The audio object is then recorded by building a representative object based on all its instances and their associated spikes, as further detailed in section 6.2. Once recorded, the instances of the object are removed from the

list and the process repeats for the next object. When no objects remain in the list, the remaining spikes from the spikegram are recorded as residual, as explained in section 6.3, and the process terminates.

6.2 Audio Object

The goal is to record each extracted object using a representative object and a set of anchor points. Each anchor point represents the location of an instance of the object. The representative object itself is purely based on the relative expressibility of the spikes associated with the other object constituents involved in each instance as a function of the spikes associated with their respective anchor points. Section 6.2.1 details how the anchor points are selected while section 6.2.2 addresses the computation of the representative object.

Each spike in the spikegram has multiple components, as mentioned in chapter 2. Depending on the spikegram experiment presented in chapter 8, not all components of each spike are always taken into consideration when forming the graph used in the graph theoretic approach of this thesis. As such, the recording of the extracted objects does not always involve all components of the spikes associated with the instances. When this happens, the spike components which are included when forming the graph are known as the primary components while those which are not included are referred to as side components. Section 6.2.3 addresses the recording of the side components.

6.2.1 Anchor Points

The anchor points must be selected such that, for each instance, the primary components of the spikes associated with all other object constituents are directly expressible as a function of the primary components of the spike associated with the anchor point chosen for the instance. It is also important that the same object constituent be chosen as the respective anchor point for each instance (i.e. either the first constituent of each instance, the second constituent of each instance, etc.). This ensures that the function relating a particular primary component of the spikes associated with each object constituent in an instance to the the same primary component of the spike associated with the anchor point of the instance is the same for all instances. In a graph context, a vertex is directly expressible as a function of an anchor point if an edge directly links both elements. Since, the extracted objects can only be stars, only the first object constituent in an

object instance is directly connected to all other constituents of the instance via an edge. As such, the first object constituents o_1 of the instances are chosen to be the anchor points, as shown in the example of figure 6.2. However, the anchor points themselves are not recorded. It is rather the primary components of the spikes associated with the anchor points that are recorded as is.

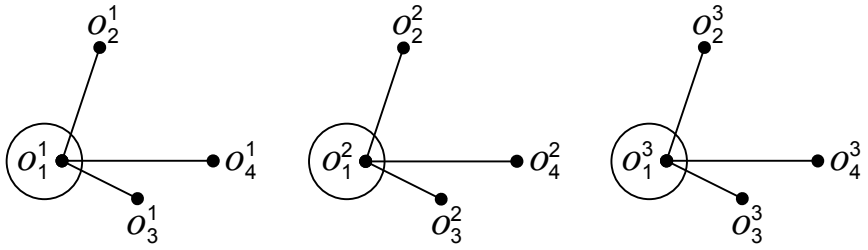


Figure 6.2: Anchor Point Selection

6.2.2 Relative Expressibility

Once the primary components of the spikes associated with the anchor points for the extracted object have been recorded, the primary components of the spikes associated with the other object constituents must be expressed as a function of the former. This is performed using relationships. Each relationship is obtained via a recording function ψ. Given a particular primary component q of the spike $s_q^{o_1^i}$ corresponding to the anchor point of an instance i and the same primary component of the spike $s_q^{o_j^i}$ associated with another object constituent j of the same instance, the recording function outputs the relationship between both inputs. The recording function varies for each primary component. The functions are also experiment-specific and are thus further described in chapter 8. The goal is to have the relationship between a primary component of the spike associated with a particular object constituent and the same component of the spike associated with its respective anchor point be the same for all instances; here onwards referred to as a representative relationship δ. This is because, in a graph context, the edge linking the vertices of both object constituents has the same label in all instances of the object, thereby implying the same relationship between the two constituents in all instances. The representative relationship is obtained by taking the arithmetic average amongst all recording functions involving a particular primary component of the spike associated with the anchor point of an instance and the same primary component of the

spike associated with the object constituent of the same instance, for all instances, as shown in equation 6.1 when computing the representative relationship for the qth spike component of the jth object constituent.

$$\delta_q^j = \frac{1}{r}\sum_{i=1}^{r}\psi_q\left(s_q^{o_1^i}, s_q^{o_j^i}\right) \tag{6.1}$$

By taking the arithmetic average, the representative relationship is guaranteed to have a value which lies within the extremes of all the values reported by the recording functions involved in the sum of equation 6.1. This ensures that the quality guarantee of the clustering algorithm in chapter 4 is preserved since the edges linking the vertices of both object constituents involved in equation 6.1 had the same labels in all instances to begin with (i.e. belonged to the same quality thresholded cluster). Here, it is assumed that the recording functions output the same relationships which are used when forming the components of the feature vectors subjected to the thresholds in the clustering algorithm.

The representative relationships for all primary components of spikes associated with all object constituents in the object other than the anchor point constituents, are recorded together as the representative object.

6.2.3 Side Components for Object Spikes

The side components of the spikes involved in the extracted objects are recorded as is, as opposed to the primary spike components which are recorded using the representative object and anchor points notation, described in the two previous sections. Thus, the side spike components form a part of the residual along with the residual spikes which do not belong to any objects.

6.3 Residual Spikes

The residual spikes are those from the spikegram which are not associated with any of the extracted objects. They can be easily identified as those which remain in the spikegram once all extracted objects have been recorded. Just as for the spikes involved in the objects, the residual spikes have multiple components. However, the difference here is that all components for each of these spikes are recorded as is, in contrast to the object spikes where the primary components are recorded using relationships.

Chapter 7

Synthesis

7.1 Introduction

Obtaining a reconstructed version of the input audio involves two steps: the reconstruction of the spikegram based on the extracted audio objects and the residual followed by the reconstruction of the audio waveform using the reconstructed spikegram. Section 7.2 details the first of these two steps while section 7.3 addresses the second.

7.2 Time-Frequency Reconstruction

To reconstruct the spikegram, the audio objects and residual are first separated into two parts. The first part, known as the residual, contains all components of the residual spikes in the spikegram and the side components of the spikes associated with the objects. The second part, known as the list of objects, contains the primary components of the spikes associated with the anchor points and the representative relationships required to rebuild the primary components of the other spikes associated with the instances of each audio object. The reconstructed version of the spikegram is then obtained as shown in figure 7.1. The primary components of the spikes corresponding to the anchor points of the first audio object, along with the corresponding representative relationships, are retrieved from the list of objects. The primary components of the spikes corresponding to the other constituents of the object instances are then obtained using a reconstruction function ζ. Given a particular primary component q of the spike $\hat{s}_q^{o_1^i}$ corresponding to the anchor point of an instance i and the same primary component of the representative relationship $\hat{\delta}_q^j$ associated with another object constituent j, the reconstruction function outputs the

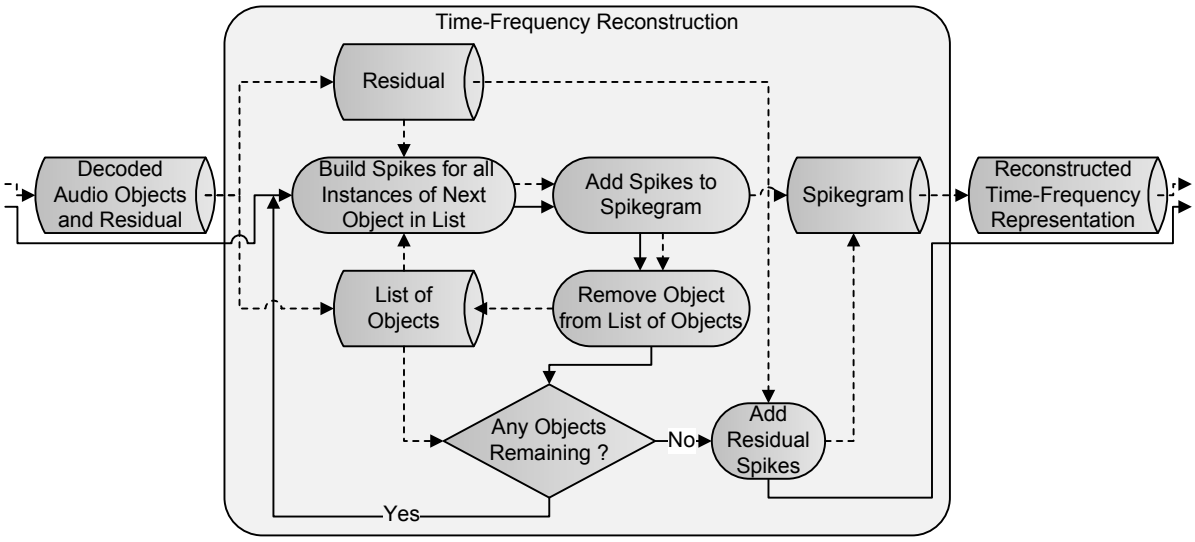


Figure 7.1: Time-Frequency Reconstruction Block

corresponding primary component of the spike $\hat{s}_q^{o_j^i}$ associated with the constituent of the representative relationship for the instance represented by the anchor point. An example of this is shown in equation 7.1 when computing the qth component of the spike associated with the jth object constituent of the ith instance.

$$\hat{s}_q^{o_j^i} = \zeta_q\left(\hat{s}_q^{o_1^i}, \hat{\delta}_q^j\right) \tag{7.1}$$

It should be noted that, as discussed in chapter 6, a primary component of the representative relationship corresponding to a particular object constituent is the same for all instances of the object. The reconstruction function varies for each primary component and has a corresponding recording function in each case. These functions are thus also further described in chapter 8. Once all primary components of all spikes involved in the instances of an object are obtained, either directly as the associated anchor points or using a reconstruction function, the corresponding side components are fetched from the residual to complete each spike. The above process then repeats for the next object in the list by using the primary components of the spikes associated with its anchor points and the corresponding representative relationships.

Once the primary and side components of the spikes associated with all objects have been placed in the spikegram, the residual spikes are added to bring the final number of spikes in the reconstructed spikegram to that of the original spikegram.

7.3 Signal Reconstruction

The audio signal waveform is obtained from the reconstructed spikegram using the procedure shown in figure 7.2. It should be noted that the dictionary of kernels used in

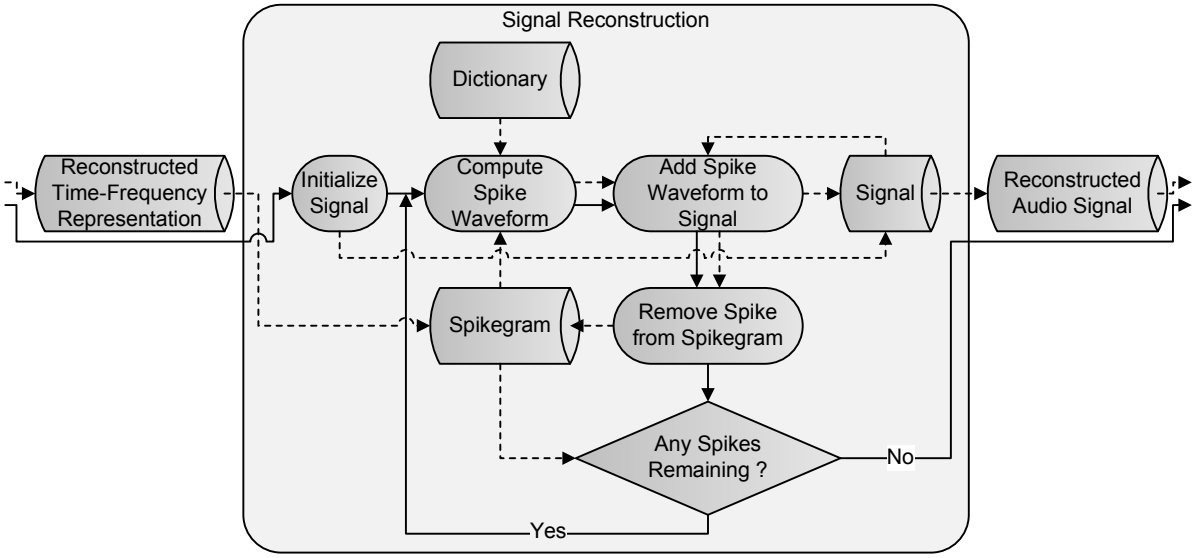


Figure 7.2: Signal Reconstruction Block

figure 7.2 is the same which was used during the original signal decomposition in chapter 2. The signal reconstruction begins with an all zero waveform for the audio signal $\hat{\mathbf{x}}^1 = \mathbf{0}$. The kernel waveform corresponding to the frequency channel of the first spike (i.e. $i = 1$) is then multiplied by the magnitude and phase of the spike and added to the audio signal waveform at the time instance indicated by the spike, as per equation 7.2. In equation 7.2, it is assumed that kernels with purely real carrier signals are employed thereby restricting the phase of the spike to either 0 or π. The resulting signal is used to add the next spike (i.e. $i + 1$) from the spikegram.

$$\hat{x}^{i+1}[n] = \begin{cases} \hat{x}^i[n] + \hat{s}^i_{mag} \exp^{j\hat{s}^i_{ang}} g_{\hat{s}^i_{chan}}[n - \hat{s}^i_{pos}], & \hat{s}^i_{pos} \leq n \leq \hat{s}^i_{pos} + M - 1 \\ \hat{x}^i[n], & \text{otherwise} \end{cases} \tag{7.2}$$

This process is repeated until no spikes remain in the spikegram. At this point, the resulting signal is the reconstructed audio wavfeorm.

Chapter 8

Experiment Design

8.1 Introduction

The framework outlined in chapters 3 through 7 is developed based on theoretical motivations. This chapter acts a transition point from these motivations to the applied world of audio. More specifically, an experiment making use of the spikegram introduced in chapter 2, is now designed to test the performance of the graph theoretic framework outlined in chapters 3 through 7 on actual audio excerpts. The following sections address the spikegram-specific aspects which must be set for the proposed experiment. Sections 8.2, 8.3, and 8.4 respectively describe the edge feature vectors, the pruning limit, and the threshold functions to be used in the graph formation process of chapter 4. The recording and reconstruction functions used in chapters 6 and 7 are then described in section 8.5. Lastly, the extraction cost function used in chapter 5 is described in section 8.6.

8.2 Edge Description

In the proposed experiment, objects are found solely based on the time instances and frequency channels of the spikes. The original spike magnitudes and phases of all spikes are sent either as side components of object spikes or as part of residual spikes. Furthermore in this problem type, objects whose instances are frequency shifted are not permitted. Only objects whose instances are located at different time instances are allowed. As such, the feature vector of each edge for this type of problem, as utilized in chapter 4, is characterized by the frequency channels of the spikes associated with both vertices and

the difference between their time instances, as shown in equation 8.1.

$$\mathbf{z}^{e_{(i,j)}} = \langle z_1, z_2, z_3 \rangle = \left\langle s^{v_i}_{chan}, s^{v_j}_{chan}, s^{v_j}_{pos} - s^{v_i}_{pos} \right\rangle \qquad i < j \tag{8.1}$$

8.3 Pruning Limit

This experiment makes use of a pruning limit during the edge formation process detailed in chapter 4. The pruning limit is applied to the time instance component of the spikes associated with the vertices of any edge. More precisely, an edge is placed between two vertices only if the difference between the time instances of their associated spikes is less than or equal to a limit N_{prune}, as shown in equation 8.2.

$$s^{v_j}_{pos} - s^{v_i}_{pos} \leq N_{prune} \tag{8.2}$$

The decision to apply a pruning limit to the time instance component of the spikes is motivated by the fact that the time duration of an object instance does not typically exceed a predefined limit. Beyond the limit, there is nothing to suggest that the spikes forming the object instance have any mutual acoustic coherence (i.e. that the human ear processes the spikes jointly as a single sound). The actual value chosen for N_{prune} is reported in chapter 9. It should be noted that for correctness, the pruning limit must be a multiple of the hop size N_{hop} since it is the hop size which decides on the indexable time instances in the spikegram, as mentioned in chapter 2.

Lastly, it should be noted that the pruning limit does not place any restriction as to where the instances of an object can occur in the spikegram. It only limits the time span of the instances.

8.4 Threshold Functions

Each of the three components defined in the feature vector for this experiment requires a threshold function. As mentioned in section 8.2, frequency shifts amongst object instances are not allowed in the proposed experiment. It is for this reason that the frequency components of the spikes associated with the vertices involved in the feature vector of section 8.2 are specified as two distinct components of the feature vector rather than using there difference as a single component. This may still mean however that the frequency components of the feature vector (i.e. the first two components of the vector) could be granted a certain amount of frequency tolerance respectively, as specified

by the threshold functions of the two components. Having said this, the goal of the proposed experiment is to have no tolerated frequency movement whatsoever. As such, the threshold functions for both components are set to zero, as shown in equation 8.3.

$$T_1 = T_2 = 0 \tag{8.3}$$

The third and last threshold function is responsible for the maximum allowable absolute displacement when comparing the time difference between the spikes associated with the two vertices in the feature vector of section 8.2 (i.e. the third component of the feature vector) to the same component in another feature vector. A key factor here is to recall that, as per the discussions in chapter 2, spikes have different effective lengths M_{eff} and center frequencies f_h based on the frequency channel in which they reside. As such, unlike the first two threshold functions which are constant, this third threshold varies based on the frequency channels involved in the feature vector, as shown in equation 8.4. In equation 8.4, μ_{max} represents the maximum allowable kernel displacement in a particular frequency channel for a given quality. Its obtention is further discussed in chapter 9.

$$T_3\left(s_{chan}^{v_j}\right) \quad = \quad \mu_{max}\left(s_{chan}^{v_j}\right) \tag{8.4}$$

From equation 8.4, it can be seen that the maximum allowable displacement is governed by the frequency channel belonging to the spike associated with the rightmost vertex of the two vertices involved in the feature vector, as per the vector definition in equation 8.1. The reasoning behind this is as follows. Assuming that two edges which have the same label (i.e. are in the same cluster) end up belonging to two different instances of the same discovered frequent star, then the leftmost vertex of each edge ends up being the anchor point of each respective instance. Seeing as the spikes associated with the anchor points do not move in the reconstructed spikegram, it is the spikes associated with the other two vertices (i.e. the rightmost vertex in each edge) which are to absorb all of the displacement caused by the usage of a representative relationship in their reconstruction. This can be illustrated by overlapping both vertices identified as anchor points to create a single reference point. In reality, the edges still occur at two distinct places in the graph where they do not overlap whatsoever. In this illustrative setting, the threshold function of equation 8.4 is designed such that the absolute time displacement, now absorbed by the two spikes not chosen as anchor points, not exceed the maximum allowable kernel displacement of the spike belonging to the feature vector with respect to which the threshold is being measured (i.e. the starting point of a candidate cluster), as

shown in figure 8.1 where the threshold is measured with respect to the spike in the left edge. In other words, if the two spikes and their associated time differences with respect

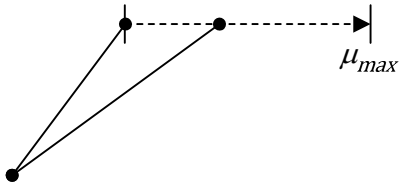


Figure 8.1: Maximum Allowable Displacement Threshold Condition

to their respective anchor points were to be swapped in the spikegram and repositioned based on each others anchor points (i.e. the spike of the second edge using the anchor point of the first edge and the spike of the first edge using the anchor point of the second edge), each newly positioned spike would not be displaced by more than the maximum allowable time shift with regards to the original time position of the other spike. Of course this represents the extreme displacement case, the new positioning of both spikes based on their respective anchor points is actually equal to the arithmetic average of both original time differences, as per the definition of the representative relationship in chapter 6. In this case, the displacement of a spike is actually equal to the difference between the original time difference of the spike and the average of the time differences. This displacement is guaranteed to be less than the difference between the two extreme time differences involved in the arithmetic average. If the difference between the two extreme differences is guaranteed to respect the maximum allowable time displacement, as is the imposed case here, then the difference between a time difference and the average time difference (i.e. the arithmetic average of the time differences) is also guaranteed to be less than the maximum allowable time displacement. It should also be noted that the maximum allowable displacement is symmetric and its application is therefore irrespective of whether the representative relationship causes a spike to move towards the left with regards to its original time position or towards the right.

Within the context of the QT clustering algorithm, having such a third threshold function is possible in this case since the component on which it depends (i.e. the frequency component of the second spike associated with the feature vectors) is also involved in the QT clustering algorithm as the second components of the candidate clusters and has the same value for all feature vectors in the candidate cluster. This therefore makes it constant and thus makes the third threshold function constant when evaluated inside

a candidate cluster formed after only considering the first two components of vectors in the hierarchical candidate cluster generation scheme of chapter 4.

8.5 Recording and Reconstruction Functions

As mentioned in chapter 6, the recording functions preserve the quality guarantee of the clustering algorithm as long as they output the same relationships which are used when forming the components of the feature vectors for the clustering algorithm.

The recording function used for the frequency component of object spikes, as shown in equation 8.5 for two inputs a and b, outputs the relationship between the inputs as a difference. This is not the same relationship that is used for the components concerned with the frequencies of the spikes (i.e. the first and second components) in the feature vector of equation 8.1. In fact, the frequency components of the spikes associated with the vertices involved in the feature vector are specified as two distinct components of the feature vector rather than using there difference as a single component. As such, the thresholds T_1 and T_2 are directly applied to the components concerned with the frequencies of the spikes during the QT clustering algorithm. Nonetheless, in this special case, the difference can still be used as the recording function because both thresholds T_1 and T_2 are set to zero. The reasoning is as follows: if two components are respectively guaranteed to be the same for all feature vectors in one cluster as per the thresholds T_1 and T_2 which are both set to zero, then their difference is also guaranteed to be same from all feature vectors in the same cluster.

$$\psi_{chan}(a, b) \quad = \quad b - a \tag{8.5}$$

In the case of the time instance component of object spikes, the recording function of equation 8.6 outputs the same relationship (i.e. the difference between the time instances) as used for the component concerned with the time instances of the spikes (i.e. the third component) in the feature vector of equation 8.1.

$$\psi_{pos}(a, b) \quad = \quad b - a \tag{8.6}$$

The reconstruction functions used for the frequency and time components of the spikes in the proposed experiment are simply the dual of the respective recording functions. Since both recording functions take the difference between two inputs to obtain a relationship, both reconstruction functions output the sum of two inputs a and b, as

shown in equations 8.7 and 8.8. In the reconstruction functions, the input b represents the relationship outputted by the respective recording function in each case.

$$\zeta_{chan}(a, b) = a + b \tag{8.7}$$

$$\zeta_{pos}(a, b) = a + b \tag{8.8}$$

8.6 Cost Function

The extraction cost function is based on bit costs. This is done because not all components involved in the cost function have the same bit weight and it would be a crude approximation to assume otherwise. When designing the bit cost of each component used in the cost function, a fixed length bit computation scheme is employed. As such, the bit cost of a component is always governed by the total number of possible values which the component in question can take on. For example, if a component can take on a maximum of 64 values then the number of bits needed to represent any of these values (i.e. the bit cost) is the same for each value and is computed as $\log_2(64)$. Variable length bit computation schemes often lead to lower bit costs. However, these schemes requires a priori knowledge about the frequency of the values taken on by the component for which the bit cost is being computed. This cannot be done in this case since the extraction of frequent stars is an iterative process. Every time a frequent star is extracted, the frequency of all component values involved in its bit cost computation along with the frequency of the component values involved in the residual changes. As such, it is impossible to know the frequency of the components and thus derive a variable length bit scheme until all frequent stars have been extracted. But the problem is that this variable length scheme would be needed to compute the bit cost of a frequent star and based on this cost decide which frequent stars should be extracted in the first place. This is similar to the "chicken and the egg" problem, not knowing which one comes first. The use of a fixed length scheme breaks this cycle and thus fixes the problem as it does not depend on the frequency of the component values. Of course the usage of this fixed length scheme means that the bit cost is suboptimal. A variable length bit scheme would have to be employed once all objects have been extracted to yield an optimal final bit rate. However, this would fall within the realm of an encoding module if the proposed framework of this thesis was used in an end-to-end coder. As mentioned in chapter 1, the focus of this thesis is on the audio object extraction process and not on the encoding of

the objects. None of the components for which a cost is computed are actually encoded as will become clear in chapter 9. The bit costs are simply the anticipated number of bits if the components were to be encoded. Furthermore, the framework is not evaluated by providing a final bit rate but rather by providing a compression gain as will also be further explained in chapter 9. As such, the development of the encoding module, if the framework was used in an end-to-end coder, is not addressed in this thesis. The usage of a fixed length bit scheme is a safe and conservative estimate since other schemes, eventually employed in an end-to-end coder, could only do better as mentioned above.

The extraction cost function is based on six factors. Assuming that the frequent star for which the extraction cost is being computed was to become the extracted object, the first factor to take into account is the bit cost attributed to the recording function used for the frequency component of object spikes. Based on its description in section 8.5, it is known that this recording function returns the difference between two inputs. Furthermore, since the function handles the frequency component of object spikes, these inputs are guaranteed to be the indices of frequency channels. As such, the total number of possible values which can be returned as the difference by the function is equal to the number of frequency channels $|H|$ (i.e. all possible absolute differences between any two channels indices) in addition to one of two possible states (i.e. a positive or negative sign) indicating the polarity of the difference. Based on these two observations, the bit cost attributed to the recording function used for the frequency component of object spikes is defined as per equation 8.9.

$$C_{\psi_{chan}} = \lceil \log_2 (|H|) \rceil + \log_2 (2) \tag{8.9}$$

The second factor to take into account is the bit cost attributed to the recording function used for the time component of object spikes. Based on its description in section 8.5, it is known that this recording function also returns the difference between two inputs. Furthermore, since the function handles the time component of object spikes, these inputs are guaranteed to be time instances. Also noting that the time duration of an object instance cannot exceed the pruning limit defined in section 8.3, the total number of possible values which can be returned as the difference by the function is equal to the ratio of the pruning limit N_{prune} over the hop size N_{hop} (i.e. the total number of time instances within the pruning limit). Since the vertices corresponding to the object spikes are ordered from left to right in the graph by convention, as mentioned in chapter 3, this largest difference will always be positive therefore not require any bits to address its polarity. Based on these two observations, the bit cost attributed to the recording

function used for the time component of object spikes is defined as per equation 8.10.

$$C_{\psi_{pos}} = \left\lceil \log_2 \left(\frac{N_{prune}}{N_{hop}} \right) \right\rceil \tag{8.10}$$

The next four factors to be taken into account for the extraction cost functions relate to the components of the spikes which will either be used as object anchor points if the frequent star for which the extraction cost is being computed was to become the extracted object or to represent residual, whether for the frequent star (i.e. the side components of the object) or for the remaining vertices in the graph (i.e. the remaining spikes). First is the frequency component of any spike. The total number of possible values for this component is equal to the number of frequency channels $|H|$. Based on this observation, the bit cost attributed to the frequency component of any spike is defined as per equation 8.11.

$$C_{s_{chan}} = \lceil \log_2 (|H|) \rceil \tag{8.11}$$

For its part, the total number of possible values which can be attributed to the time component of any spike is equal to the ratio of the signal length N over the hop size N_{hop} (i.e. the total number of time instances). Based on this observation, the bit cost attributed to the time component of any spike is defined as per equation 8.12.

$$C_{s_{pos}} = \left\lceil \log_2 \left(\frac{N}{N_{hop}} \right) \right\rceil \tag{8.12}$$

The magnitude differs from the other spike components in that it is continuous. As such, it must first be quantized into a finite number Γ of discrete levels before its bit cost can be calculated. This number of levels is reported in chapter 9. However, given a number of levels, the total number of values which the discretized magnitude can take on is equal to this number of levels. From this observation, the bit cost attributed to the magnitude component of any spike is defined as per equation 8.13.

$$C_{s_{mag}} = \lceil \log_2 (\Gamma) \rceil \tag{8.13}$$

Lastly the phase component of any spike is restricted to two values (i.e. 0 or π) as mentioned in chapter 2. Based on this observation, the bit cost attributed to the phase component of any spike is defined as per equation 8.14.

$$C_{s_{ang}} = \log_2 (2) \tag{8.14}$$

The six costs defined above are now combined into the extraction cost function, as shown in equation 8.15.

$$\begin{aligned} C(Q,G) &= (k-1)\left(C_{\psi_{chan}} + C_{\psi_{pos}}\right) \\ &+ r\left(C_{s_{chan}} + C_{s_{pos}}\right) \\ &+ rk\left(C_{s_{mag}} + C_{s_{ang}}\right) \\ &+ (|V| - rk)\left(C_{s_{chan}} + C_{s_{pos}} + C_{s_{mag}} + C_{s_{ang}}\right) \end{aligned} \tag{8.15}$$

Given a frequent star and a graph in which this frequent star resides, the extraction cost of the frequent star is computed based on four terms. The first term in equation 8.15 represents the combined cost associated with the recording functions if the frequent star was to be extracted as an object. Since the frequent star has a size of k, its extraction will require the cost of $k-1$ sets of recording functions (i.e. the number of edges from the anchor point to all other vertices in an instance of the object as discussed in chapter 6). The second term represents the cost associated with the anchor points if the frequent star was to be extracted as an object. It should be noted that only the frequency and time components of the spikes used as the anchor points are considered in this term since these are the primary components of the spikes (i.e. the magnitude and phase components are side components for this experiment). Since the frequent star has a frequency of r, its extraction will require the cost of r sets of primary components (i.e. the number of anchor points required to address all instances of the object as discussed in chapter 6). The third term represents the side components of the spikes associated with the vertices of the frequent star. Recall that these side components are sent as is (i.e. one set of side components per vertex). Since the total number of vertices covered by the frequent star is rk, these vertices will require a cost of rk sets of side components. Lastly, the fourth term in equation 8.15 represents the cost associated with the remaining vertices in the graph. Since the total number of vertices in the graph is $|V|$, the remaining vertices will require a cost of $|V| - rk$ sets of spike components.

Chapter 9

Performance

9.1 Introduction

The performance of the framework proposed in this thesis based on the experiment designed in chapter 8 is now discussed. To do so, benchmarks indicating how the required tests are conducted are first described in section 9.2. The metrics used to evaluate the performance are then detailed in section 9.3. Lastly, section 9.4 presents the obtained results along with a discussion of their significance.

9.2 Setup

The main tests conducted in this chapter make use of a unique testing environment, known as the testbed. It is presented in section 9.2.1 along with the reasoning behind its creation. This testbed requires an audio file as input. However, audio, by definition, spans a wide range of sounds. As such, it is desirable to examine the performance of the framework when presented with different sounds from this wide ranging spectrum. An audio corpus composed of a finite number of audio files - one for each test - is thus created. The corpus is described in section 9.2.2. Lastly, other settings used to perform the tests are described in section 9.2.3.

9.2.1 Testbed

A key aspect factored into the proposed testbed is the limitation imposed on the graph size which can be processed by the subgraph extraction process of chapter 5. As men-

tioned in chapter 3, the size $|V|$ of the original graph in the framework is equal to the number of spikes $|S|$ in the spikegram. Informal trials have shown that the subgraph extraction module of chapter 5 is computationally expensive when processing large graphs. It should be noted that exact algorithm execution times are not discussed in this thesis seeing as the current aim of the implementation is purely a proof of concept. To allow for smaller graphs all the while maintaining a large input spikegram, the testbed makes use of a pipeline procedure. More precisely the pipeline procedure is applied during the star extraction process such that the number of spikes during the frequent star search is limited by the size of the pipeline. The procedure begins by filling the pipeline with the vertices corresponding to the first spikes from the spikegram until the pipeline is full (i.e. the number of spikes in the pipeline is equal to the pipeline size). The frequent star search is then executed on these vertices. When the frequent star search completes, the vertices associated with the optimal frequent star (i.e. the one with the minimal extraction cost) remain in the pipeline and the other vertices are removed and set aside in a buffer. The vertices corresponding to the next spikes from the spikegram (i.e. from where the pipeline stopped filling previously and onwards) are then added to the partially filled pipeline until it is filled to capacity and the frequent star search process is repeated. If the frequent star search reports an optimal frequent star which has a lower extraction cost than the optimal frequent star found during the previous pipeline run, then the vertices associated to this new optimal frequent star are the ones which remain in the pipeline. Otherwise, the vertices associated with the optimal frequent star found during the previous pipeline run are the ones which remain in the pipeline. It should be noted that in the second case, the vertices associated with the optimal frequent star found during the previous pipeline run are guaranteed to be in the current pipeline since they are the ones which remained in the pipeline when the previous pipeline run completed. Again, all other vertices are removed from the pipeline and added to the buffer. This process repeats until all spikes from the spikegram have been considered. The vertices associated with the optimal frequent star at the end of the process become the recorded object. The vertices which are placed in the buffer are then resorted according to a desired spike order and the pipeline procedure re-operates on them to find a second object. The pipeline procedure runs repeatedly as such until the stopping criterion of the subgraph extraction process of chapter 5 is met. It should be recalled here that prior to even extracting any subgraph, the vertices corresponding to all spikes in the spikegram are indexed based on the left-to-right temporal location of the spikes. As such, regardless of the order in which the spikes themselves are sorted, the subset of vertices which are

taken into the pipeline are always resorted based on their indices. This guarantees that, when working with a subset of vertices (i.e. a pipeline), the frequent star search can still assume that vertex indices are indexed from left to right in increasing order.

Of course, it is the order of the spikes themselves which determine which vertices are first read into the pipeline and so on. The question thus becomes: how should the spikes themselves be sorted before executing the pipeline procedure? Since the goal of the proposed framework is to find objects which occur at different time instances in the spikegram, it is in the interest of the testbed to preserve the time axis N and frequency axis $|H|$ of the original spikegram when forming a pipeline. As such, the spikes are sorted based on amplitude. When forming the spikegram in chapter 2, the spikes are extracted in order of decreasing amplitude seeing as the kernel which correlates most with the signal (i.e. the spike with greatest amplitude) is extracted first, and so on. This order is simply preserved for the pipeline procedure. As such, the first pipeline is formed by taking the first extracted spikes when forming the spikegram, and so on.

The reasoning behind preserving the original order of spike extraction is the belief that the objects to be extracted are rooted by vertices corresponding to high amplitude spikes since these spikes represent high energy areas in the spikegram. By performing the first pipeline run with the vertices corresponding to these spikes, the vertices which are kept in the pipeline for the following pipeline run are those corresponding to a frequent star containing high amplitude spikes. The next run can then expand this frequent star with the newly added vertices to the pipeline which correspond to lower amplitude spikes. Thus, although the pipeline procedure is a heuristic mechanism which reduces the search space by not considering the vertices of all spikes in the spikegram at once, processing the spikes in their original spikegram extraction order minimizes the approximative effects of this heuristic due to the above mentioned belief. It should be noted that the procedure still allows for the optimal frequent star in a later pipeline run to be formed from entirely new vertices just added to the pipeline rather than just expanding off those retained from the previous pipeline run. After each recorded object, the vertices in the remaining buffer are also always re-sorted such that their corresponding spikes retain the original order of extraction when the spikegram was formed. Lastly, it should be noted that the pipeline procedure is presented in this chapter rather than chapter 5 since the above reasoning which led to its development is specific to the spikegram and not subgraph extraction in general. For the tests in this chapter, the pipeline size is set to 1000 since informal tests have shown that beyond this limit the subgraph extraction process of chapter 5 slows down significantly.

The testbed itself operates as shown in figure 9.1. The testbed is first presented with an input audio file, here onwards referred to as audio file A. The spikegram of this audio file A is then obtained via the signal decomposition process proposed in chapter 2. At this point, an intermediate step is performed where the spikegram is reconstructed into an audio signal, here onwards referred to as audio file B, via the signal reconstruction procedure outline in chapter 7. Since the signal decomposition process is lossy, having audio file B allows for an intermediate reference point when evaluating the performance to examine whether a loss of audio quality is caused by the front end signal decompositon process or by the proposed framework which follows in the testbed. Once this intermediate step is completed, the testing procedure continues with the spikegram of audio file A. Given the spikegram as input, the object extraction framework proposed in chapters 3 to 6 converts the spikegram into a set of objects and residual all the while employing the pipeline procedure outlined above during subgraph extraction portion of the framework. As mentioned in chapter 1, the focus of this thesis is on the object extraction process rather than proposing a full end-to-end audio coder. As such, the encoding and decoding modules of figure 1.2 are not addressed in this thesis. The objects and residual are thus immediately reconstructed into a spikegram via the time-frequency reconstruction procedure outline in chapter 7. This spikegram is then reconstructed into a final audio file, known as file C.

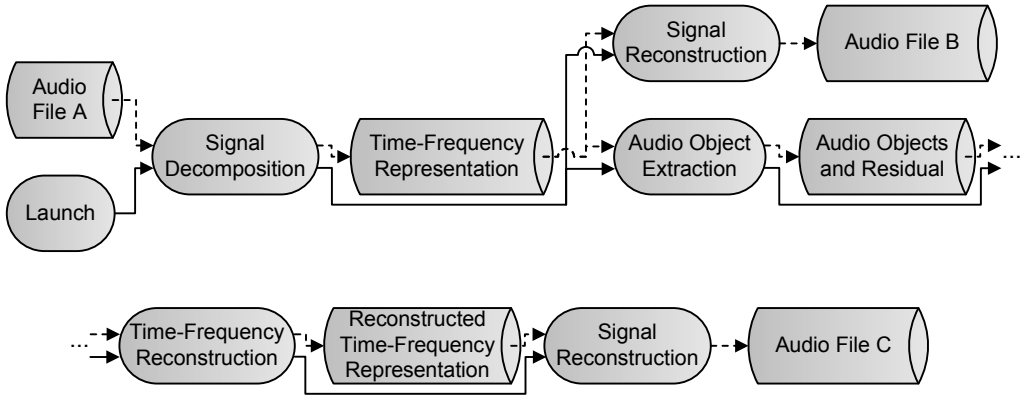


Figure 9.1: Testbed

9.2.2 Corpus

The corpus used for the tests contains five audio excerpts. The five sounds are chosen such that they span a wide range of audio genres. They are that of a castanet, a vibraphone, a female speech utterance in english, a male speech utterance in german, and a piano. Since the longterm goal is to eventually use the proposed framework of this thesis in a high-quality wideband audio coder, the excerpts are produced using a sampling frequency of 44.1 kilohertz (ie. $F_s = 44100$). The length in samples of each excerpt is shown in table 9.1.

Audio	Length (samples)
Castanet	100000
Vibraphone	240000
Female Speech	308000
Male Speech	328000
Piano	332000

Table 9.1: Audio File Lengths

9.2.3 Other Settings

The signal decomposition process is performed using a 25 channel Gammatone kernel set (i.e. $|H| = 25$) since it is shown in [48] that using more than 25 channels does not increase the performance of the signal decomposition process significantly. All kernels have a window length of 4000 (i.e. $M = 4000$). In addition, a hop size of 1 (i.e. $N_{hop} = 1$) is employed to allow maximum temporal resolution in the spikegram.

A kernel displacement curve is computed for each kernel in the Gammatone kernel set. The kernel displacement curve offers a quantitative manner of relaying the amount of displacement of a kernel with respect to itself to the amount of energy captured from the original kernel by the displaced kernel. Each curve is obtained by taking two normalized kernels in the same channel, sliding one away from the other, and calculating the signal to noise ratio for different displacements. The definition of the signal to noise ratio is given in section 9.3. Since both kernels are identical in shape, the displacement function is symmetric (i.e. negative displacements yield the same signal to noise ratio value as positive displacements). Figure 9.2 shows the positive half of a displacement curve for a

Gammatone kernel. A key observation to be made in figure 9.2 is the periodic nature of

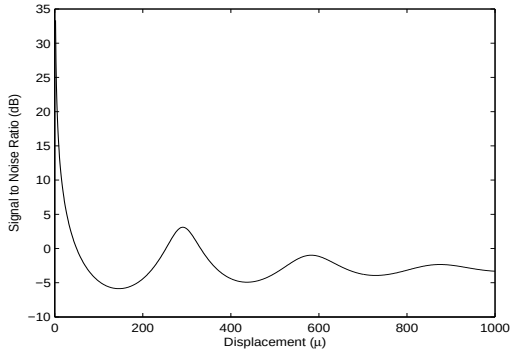


Figure 9.2: Kernel Displacement Curve for Frequency Channel $h = 2$

the displacement curve. This is because both kernels carry carrier signals as discussed in chapter 2. When the kernels are in phase, positive peaks occur in the curve, and the contrary occurs when the kernels are out of phase. It also happens that the curve has negative signal to noise ratio values. This simply means that, for these displacements, the displaced kernel actually introduces more of its own energy than the amount which it captures from the original kernel (i.e. the non-displaced kernel).

The maximum allowable kernel displacement μ_{max} required for each channel can be obtained by using the kernel displacement curve of each frequency channel and the absolute threshold of hearing. The absolute threshold of hearing quantifies hearing sensitivity at different frequencies. In truth, one should use equal-loudness contour curves to obtain a precise assessment of hearing sensitivity at different frequencies. Each equal-loudness contour curve reports hearing sensitivity at different frequencies for a given loudness level (i.e. the decibel level to which the curve is shifted). Based on the given loudness levels, the shape of each of these curves varies slightly from one curve (i.e. one decibel level) to the next. However, this variation is very small. One approximate shape can thus be used for all the curves, avoiding the need to have a slightly different curve for each loudness level. This observation combined with the fact that the absolute threshold of hearing best resembles the shapes of all equal-loudness contour curves is what motivates its usage as an approximation. By inverting the absolute threshold of hearing, an approximate assessment of in which channels the highest quality should be preserved is obtained. The most sensitive frequency channel (i.e. the channel where the quality must be most preserved) corresponds to the maximum point in this inverted threshold.

The absolute threshold of hearing as defined in [19] for a 25 channel Gammatone kernel set is shown in figure 9.3, once inverted and shifted such that the decibel level in the most sensitive frequency channel is 0 dB. To obtain a set of maximum allowable ker-

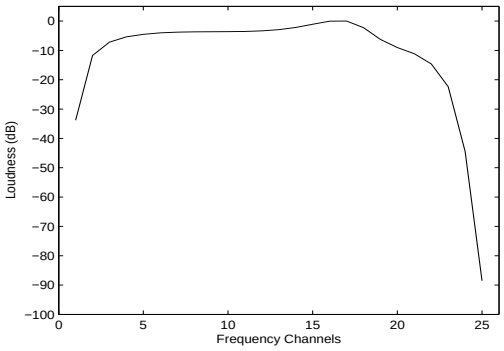


Figure 9.3: Inverted Absolute Threshold of Hearing for Gammatone Kernel Set

nel displacements, one only has to specify the desired minimum tolerable quality in the most sensitive frequency channel, which in the case of figure 9.3 corresponds to channel 17. The actual minimum tolerable quality used for the tests is reported in section 9.4. The inverted threshold is then shifted such that the decibel level in the most sensitive frequency channel corresponds to this value. The corresponding decibel values for all other channels can then be obtained from the shifted and inverted threshold of hearing. Each decibel value is then taken to the respective kernel displacement curve and the displacement value corresponding to the decibel value along the curve becomes the maximum allowable kernel displacement for the frequency channel in question. It should be noted that when reading off the maximum allowable displacement from a displacement curve, only the first monotonic decrease portion of the displacement curve (i.e. from zero displacement and outwards) is considered. If the decibel value is lower than the minimum decibel point in this portion of the curve (i.e. the point before the curve begins rising again), the displacement value corresponding to the minimum point is taken as the maximum allowable kernel displacement for the frequency channel in question. This is because, when recording an object, the representative relationship for the time component of spikes takes the average of a set of values which cannot diverge by more than this maximum allowable kernel displacement. Therefore, the difference between any value and the average, which is guaranteed to be less than or equal to this upper bound, must correspond to a decibel value (i.e. quality) along the displacement curve which is

equal or greater than the decibel value corresponding to this upper bound. The only way of insuring this, is by remaining in the monotonic portion of the curve (i.e. when moving away from a zero displacement value).

As mentioned in chapter 3, it is only when labelled edges co-occur that evidence of structure becomes apparent. As such the minimum frequent size (i.e. minimum number of vertices) during the frequent star search of chapter 5 is set to 3 (i.e. $k_{min} = 3$) to guarantee that frequent stars are composed of co-occuring edges (i.e. more than a single edge). To limit the search space of the frequent star search such that it operates in reasonable time, a maximum star size of 15 (i.e. $k_{max} = 15$) is set. The second condition required for evidence of structure is the repetition of a particular sequence of co-occuring edges in more than one place in a graph. As such, the minimum star frequency during the frequent star search is set to 2 (i.e. $r_{min} = 2$).

Lastly, the pruning limit is set to 10000 (i.e. $N_{prune} = 10000$) and the number of quantization levels for the amplitude is set 1024 (i.e. $\Gamma = 1024$).

9.3 Evaluation Methods

The performance of the framework is evaluated based on two factors: the compression which can be achieved with regards to the time-frequency representation of the input audio and the retained audio quality at various stages in the testbed. Section 9.3.1 shows how the compression is measured. For its part, section 9.3.2 presents the metrics used to measure the audio quality of the files generated in the testbed.

9.3.1 Compression

The compression achieved by the proposed framework is evaluated by using the extraction cost function defined in chapter 8 and by tracking an overall cost Λ. The overall cost tracks the cost associated with the cumulative removal of all extracted objects as they are extracted (i.e. not just the cost of the current object being extracted but also the cost of all previous extracted objects) and the corresponding residual which changes every time an additional object is extracted. The overall cost is initialized to be equal to the extraction cost of the original graph G_0 if the null object $\emptyset$ is extracted from it (i.e. $\Lambda_0 = C(\emptyset, G_0)$). The null object represents the case where no object is extracted from the graph. It is defined as having a frequency of zero (i.e. $r = 0$) and a size of one (i.e. $k = 1$) such that the three first terms (i.e. the terms responsible for the object cost) in

the extraction cost function of chapter 8 disappear meaning that the extraction function effectively only computes the cost of the input graph based on its size $|V|$. When the ith object O_i is extracted, the overall cost is updated by replacing the cost of the ith graph G_i when no object is extracted from it (i.e. represented by the null object) and replacing it by the cost of the ith graph when the ith object is extracted from it, as shown in equation 9.1. It should be noted that, as per the removal process of a frequent star detailed in chapter 5, the ith graph G_i is always equal in size to the $i-1$ graph G_{i-1} once the $i-1$ object O_{i-1} has been removed from it, thereby representing the residual. As such, with its iterative update, equation 9.1 essentially replaces the residual portion of the overall cost, the latter of which also contains the cost of previously extracted objects, by a new and smaller residual cost and the cost of a new extracted object.

$$\Lambda_i \;=\; \Lambda_{i-1} - C(\emptyset, G_i) + C(O_i, G_i) \tag{9.1}$$

This above process continues until the maximum number of objects # have been extracted. The final overall compression is then computed as one minus the ratio of the final overall cost to that of the initial overall cost, as shown in equation 9.2.

$$\text{Overall Compression} \;=\; 1 - \frac{\Lambda_{\#}}{\Lambda_0} \tag{9.2}$$

The relative compression is another quantity of interest since it reports the gain achieved with regards to the spike components which are being compressed (i.e. only the frequency and time components of the spikes). Much like the overall compression, the relative compression can be obtained as one minus the ratio of the final overall cost to that of the initial overall cost, but this time subtracting the cost of the magnitude and phase components of the spikes in the original graph G_0 from both the numerator and denominator of the ratio, as shown in equation 9.3.

$$\text{Relative Compression} \;=\; 1 - \frac{\Lambda_{\#} - |V_0|\left(C_{s_{mag}} + C_{s_{ang}}\right)}{\Lambda_0 - |V_0|\left(C_{s_{mag}} + C_{s_{ang}}\right)} \tag{9.3}$$

9.3.2 Audio Quality

Three metrics are used to evaluate the audio quality of the files in the testbed: the Signal to Noise Ratio (SNR), the Segmental Signal to Noise Ratio (SSNR), and the Perceptual Evaluation of Audio Quality (PEAQ) model. The SNR and SSNR are mathematical measures which are calculated purely based on waveform samples without taking any other knowledge into consideration. Given a reference signal $\mathbf{x}_{ref}$ and a test signal $\mathbf{x}_{test}$,

the SNR is computed as the ratio of the energy of the reference signal to the energy of the difference between the reference signal and the test signal (i.e the noise). The SNR is reported in decibels by mapping the obtained energy ratio to a logarithmic scale, as shown in equation 9.4.

$$SNR = 10\log_{10}\left(\frac{\sum_{n=0}^{N-1}(x_{ref}[n])^2}{\sum_{n=0}^{N-1}(x_{ref}[n]-x_{test}[n])^2}\right) \tag{9.4}$$

A key problem with the SNR is that it attributes more importance to high amplitude segments in the test and reference signals since the samples in these segments are considered jointly with samples from lower amplitude segments when computing equation 9.4 [49]. This leads to a measure which is reflective of the quality in high amplitude segments in the reference and test signals rather than low amplitude segments [49]. This can be misleading since discrepancies in low amplitude segments can also lead to significant quality degradation [49]. The SSNR addresses this issue by first dividing both the reference and test signals into segments of length N_{seg} such that the samples of one segment do not affect the samples of another. By doing so, discrepancies in segments containing low amplitude samples have just as much importance as those occurring in segments with high amplitude samples when computing the audio quality measure [49]. The SSNR is computed by first taking the SNR of each segment in decibels. The final value is then reported as the average of all the segment SNRs, as shown in equation 9.5.

$$SSNR = \frac{N_{seg}}{N}\sum_{i=0}^{\frac{N}{N_{seg}}-1}\min\left\{10\log_{10}\left(\frac{\sum_{n=iN_{seg}}^{(i+1)N_{seg}-1}(x_{ref}[n])^2}{\sum_{n=iN_{seg}}^{(i+1)N_{seg}-1}(x_{ref}[n]-x_{test}[n])^2}\right),\sigma\right\} \tag{9.5}$$

In equation 9.5, an upper bound σ in also placed on the SNR of any segment. If the SNR of a segment exceeds this upper bound, it is replaced by the value of the bound. This is done to prevent segments with extremely high SNRs from biasing the final SSNR upwards. The upper bound is specified in decibels and set to 40 decibels (i.e. $\sigma = 40$) for the tests conducted in this chapter. This choice is biologically motivated, since added acoustic quality corresponding decibel levels above this value is rarely noticed by the human ear. For its part, the segment length is set to the small value of 256 (i.e $N_{seg} = 256$) to enforce a large number of segments and thus increase the chances of separating low amplitude samples from high amplitude samples.

The PEAQ model is an International Telecommunication Union (ITU) standard for evaluating audio quality. Contrary to the SNR and SSNR measures, the PEAQ model not only takes into account waveforms samples when evaluating audio quality but also human

behaviour in mimicking the human auditory processing system. Given a test signal and a reference signal, the model first pre-processes the signals based on the psychoacoustic properties of the human ear. The model then sends the resulting signals through a neural network which has been trained a priori from auditory tests with humans to mimic the cognitive aspects of the human auditory processing system. Lastly, the model outputs a set of variables which map to a score ranging between 0 and -5. A score above -1 is said to be of broadcast quality. The score scale is further detailed in table 9.2. Due to its

Score Range	Perceptual Difference
0 to -1	Imperceptible
-1 to -2	Perceptible but not annoying
-2 to -3	Slightly Annoying
-3 to -4	Annoying
-4 to -5	Very Annoying

Table 9.2: PEAQ Score Scale

extensive biological foundations, it is possible for the PEAQ module to output a good score given a test and reference signal even though the corresponding SNR and SSNR are low. This is because the human auditory processing system is often insensitive to certain types of discrepancies between a reference and test signal. These aspects are factored into the PEAQ module but not into the SNR and SSNR measures.

9.4 Results and Discussion

The results from running each audio excerpt from the corpus through the testbed are first reported in section 9.4.1. Section 9.4.2 then discusses some of the observations drawn from these results.

9.4.1 Results

The signal decomposition process is run for each audio excerpt until the PEAQ score of file B with reference to file A in each case is at least -1 (i.e. broadcast quality). Table 9.3 shows the amount of spikes required to achieve such a score for each audio excerpt along with the actual PEAQ score achieved and the corresponding SNR and SSNR in each case.

Audio	Number of Spikes	SNR (dB)	SSNR (dB)	PEAQ Score
Castanet	14000	17.44	14.21	-0.902
Vibraphone	41000	38.17	27.82	-0.942
Female Speech	70000	27.22	25.84	-0.967
Male Speech	61000	31.87	25.79	-0.962
Piano	75000	36.99	33.88	-0.908

Table 9.3: Performance of Signal Decomposition

The minimum tolerable quality in the most sensitive frequency channel is also chosen such that the PEAQ score of file C with reference to file B for all audio excerpts is at least -1 (i.e. broadcast quality), as is shown later in table 9.5. From experimentation, this value was found to be 12 dB in the most sensitive frequency channel.

Table 9.4 shows the number of edges in the graphs formed from the spikegrams of the respective audio excerpts before applying the pruning limit and after its application. The same table also shows the number of clusters obtained when executing the QT clustering algorithm for each audio excerpt with the thresholds corresponding to the chosen minimum tolerable quality.

Audio	Number of Edges before Pruning	Number of Edges after Pruning	Number of Clusters
Castanet	97993000	24621047	2404539
Vibraphone	840479500	101707562	3000099
Female Speech	2449965000	184556688	3094316
Male Speech	1860469500	131262796	3111024
Piano	2812462500	198422105	2773057

Table 9.4: Performance of QT Clustering

Lastly, table 9.5 shows the complete performance of the graph theoretic framework when working on the labelled graphs of the respective spikegrams. In each case, the total number of extracted objects is reported along with the overall (O) and relative (R) compression and the SNR, SSNR, and PEAQ scores when comparing file C to file B and file C to file A. Although table 9.5 reports the quality of file C with respect to file A for completeness, emphasis is placed on the quality comparison of file C with respect to file B

Audio	Number of Objects	Compression (%)		Test/Ref Files	SNR (dB)	SSNR (dB)	PEAQ Score
Castanet	1178	O	28.67	C/B	10.41	12.18	-0.752
		R	43.00	C/A	9.60	10.04	-1.294
Vibraphone	2305	O	24.44	C/B	21.18	16.06	-0.734
		R	36.14	C/A	21.09	15.60	-1.688
Female Speech	1318	O	8.71	C/B	16.16	17.96	-0.191
		R	12.71	C/A	15.84	16.07	-1.114
Male Speech	1061	O	8.39	C/B	19.47	20.45	-0.125
		R	12.24	C/A	19.16	16.12	-1.127
Piano	1395	O	9.28	C/B	20.60	21.03	-0.175
		R	13.54	C/A	20.45	20.19	-1.343

Table 9.5: Performance of Graph Theoretic Framework

since it is on file B that the graph theoretic framework operates. When emphasizing this comparison in table 9.5, the proposed framework is capable of achieving average overall and relative compression gains of 15.90% and 23.53% respectively while maintaining an average PEAQ score of -0.395. An informal subjective evaluation of file C with respect to file B reveals that the discrepancies reflected by the PEAQ scores residing between 0 and -1 are present in the form of minors reverberations being introduced into file C for all excerpts.

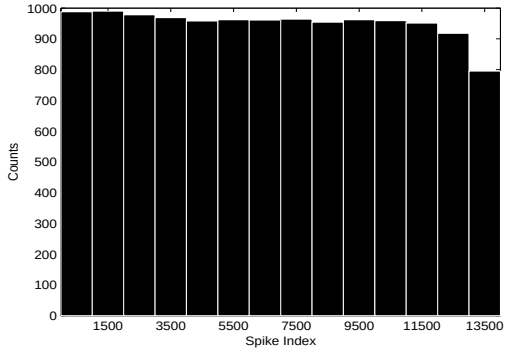
9.4.2 Discussion

A first observation from table 9.5 is that both speech files have the lowest compression gains. This is to be expected since speech, in general, contains high structural variability. In other words, each phoneme has its own unique structure which - a part from underlying harmonics if the phoneme is voiced - is not common to other phonemes unless the phoneme itself is repeated. In contrast, the castanet and vibraphone files achieve the highest compression gains since they are both very structured types of sounds. The structure of the castanet excerpt is further discussed later in this section. For its part, the vibraphone is generally a very reverberant instrument.

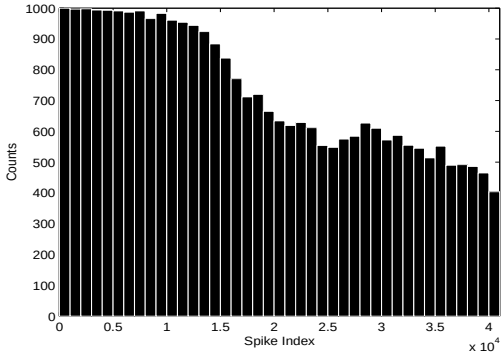
A second observation is to be made when looking at the comparisons of file C to file B for each audio excerpt in table 9.5. In these comparisons, the PEAQ score remains

high despite a low SNR and SSNR. This is due to the usage of the absolute threshold of hearing when determining the maximum allowable shift for the time thresholds of the QT clustering algorithm. As can be seen in figure 9.3, the inverted absolute threshold of hearing allows for a lower tolerable quality and hence bigger allowable shifts for low and high frequency channels. As such, kernel shifts in these channels of the reconstructed spikegrams are bigger than in mid-range frequency channels, leading to lower reconstruction quality in low and high frequency channels. The SNR and SSNR drop because they attribute equal importance to all frequency channels in their calculations. Yet the PEAQ score remains high because it factors in the fact that, as per the inverted threshold of hearing, the low reconstruction quality in low and high frequency channels is not perceptually relevant. The discrepancies between the PEAQ score and the SNR and SSNR thus show that the graph theoretic framework successfully takes advantage of these bigger shifts in low and high frequency channels to obtain greater compression gains without affecting the perceptual quality of the reconstruction as reported by PEAQ. This is exactly the goal of using the absolute threshold of hearing when determining the maximum allowable shift for the time thresholds of the QT clustering algorithm.

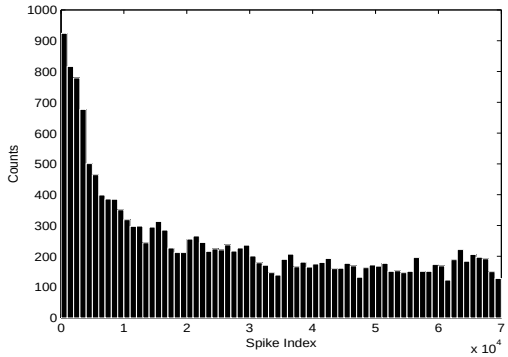
Another interesting observation is the area of the spikegram in which the spikes belonging to the objects reside. Figure 9.4 shows the distribution of the indices of these object spike amongst the total number of spikes of the original spikegram in each case. It should be noted that in this figure, the spikes are indexed in the order in which they are extracted during the signal decomposition process (i.e. spike index 1 is the first extracted spike, spike index 2 is the second extracted spike, etc.). Knowing that - as mentioned in section 9.2.1 - the spikes are extracted in order of decreasing amplitude, it is clear from figure 9.4 that most of the spikes belonging to the objects correspond to high amplitude spikes in the spikegram (i.e. the first spikes to be extracted during the signal decomposition process). Even in the castanet case, where almost all the bars in the histogram are equal because a lot of structure is extracted from the spikegram, the first bars are slightly higher than those which follow. This observation not only confirms the belief mentioned in section 9.2.1 that the objects to be extracted are rooted by vertices corresponding to high amplitude spikes but also shows that structure in an audio signal comprises most of its energy. One could argue that the usage of the pipeline procedure in the testbed biases the histograms since the first pipeline is formed by taking the first extracted spikes when forming the spikegram, and so on. However, this conclusion would be wrong as the pipeline mechanism allows for the optimal frequent star in a later pipeline run to be formed from entirely new vertices just added to the pipeline rather



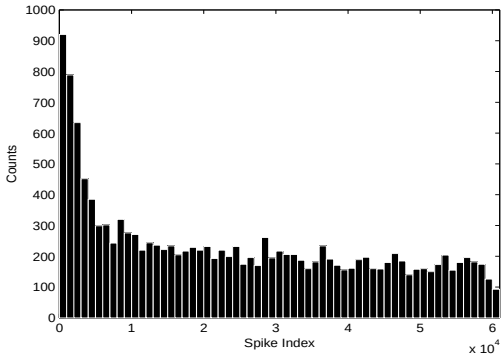
(a) Castanet



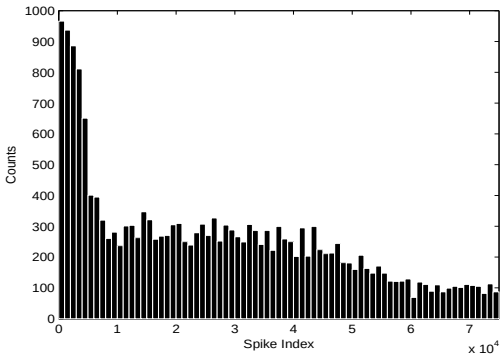
(b) Vibraphone



(c) Female Speech



(d) Male Speech



(e) Piano

Figure 9.4: Object Spike Locations in Spikegram

than just expanding off those retained from the previous pipeline run, as mentioned in section 9.2.1.

Looking at the shape of some of the extracted objects, it can also be seen that new types of objects are found. To illustrate an example, the castanet spikegram is used since it contains the least amount of spikes and is thus easiest for visualization purposes. More specifically, this spikegram contains 13 audio events, as denoted by the boxes in figure 9.5. Each event is composed of an attack, as illustrated by the vertical positioning

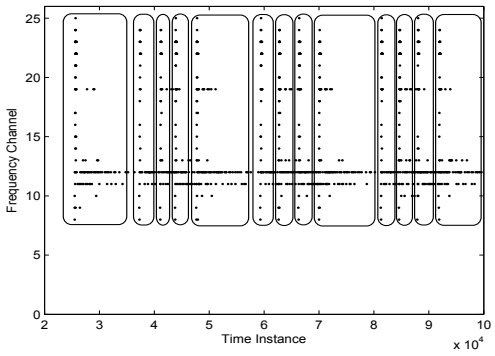


Figure 9.5: Spikegram for Castanet Audio Excerpt

of spikes inside each box, followed by a steady state, as illustrated by the horizontal positioning of spikes in each box. It should be noted that figure 9.5 shows only the first 1000 spikes of the spikegram (as per their order of extraction), such that reader can readily identify the 13 events.

The instances of one of the objects discovered in the castanet spikegram are shown in figure 9.6. Here it can be seen that the instances involve spikes from both the attack portion and steady state portion of the two events in which they are involved. As mentioned in chapter 1, current methods in industry are capable of representing the attack portion of the events with quality and the steady state portion of the events with quality. However, quality is often lost when trying to model the area between the attack portion and steady state portion as the combination of both these portions of the signal. In figure 9.6, this hybrid portion is captured as one piece. This should not be confused with a high level object which is a combinations of smaller objects. The extracted object is rather an entity of its own which bridges the attack portion of the event and the steady state portion by drawing spikes from both portions. An example of a high level object is shown later in this section for clarity.

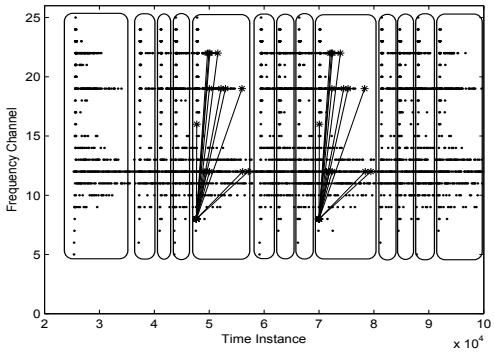


Figure 9.6: Object Discovered in Castanet Spikegram

A key observation to be made in figure 9.6 is that, from looking at the spikegram, more spikes from the two events in which the instances are involved could be added to the instances since the events appear similar beyond the spikes involved in the instances. The non-removal of some of these additional spikes as part of this object is partly due to the usage of the pipeline procedure for the testbed and the usage of an upper limit on the star size for the frequent star search, as is further discussed later in this section. More importantly, however, some of these additional spikes are not removed as part of the instances of the object because they simply do not sound the same. This very important point is what differs audio analysis from audio compression: structures which appear to be identical in the spikegram may not entirely sound similar within a certain quality guarantee. This point justifies the usage of QT clustering which has quality guarantees rather than any unsupervised learning algorithm when labelling the graph. It also illustrates the power of the graph theoretic framework: picking out only the spikes belonging to the subsets of the events which are guaranteed to sound similar within a quality guarantee rather than the entire events.

Of course some of the discovered objects can be high level objects as shown in figure 9.7. In this figure, each instance of the object involves spike from more than one castanet event. This is simply a consequence of the imposed pruning limit being too long in the specific case of the castanet audio excerpt. The pruning limit imposes a maximum instance duration but does not say anything about combining events whose combined duration is less then this maximum if the spikes selected from the combined events give a better compression gain.

Perhaps the most important observation of all can be drawn by looking at the dis-

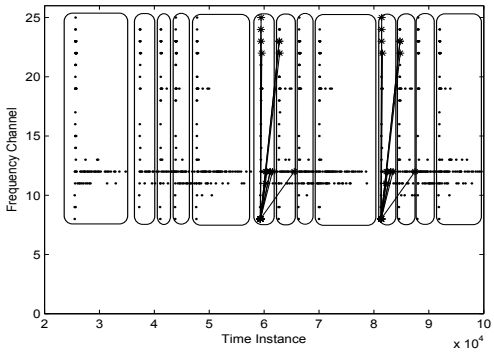
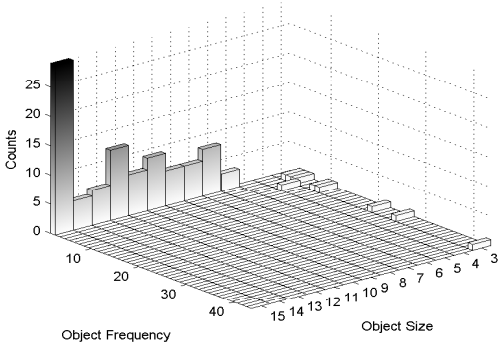


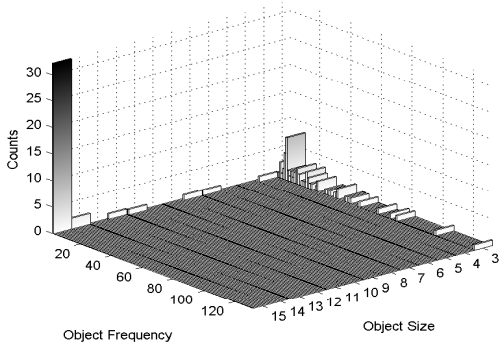
Figure 9.7: High Level Object Discovered in Castanet Spikegram

tribution of the discovered objects based on their size and frequency in the case of each audio excerpt. Figure 9.8 illustrates this well as a bivariate distribution for the first 100 objects of each excerpt in question. It should be recalled here that the subgraph extraction process of the graph theoretic framework is iterative in nature and, as a consequence, the extraction cost of each recorded object decreases as the number of extracted objects increases. As such, only the first 100 objects are taken into account for the distributions of figure 9.8 since they are the objects which contribute most to the compression gains reported in table 9.5. As can be seen from figure 9.8, almost all of the objects discovered in each case are either very small and have a wide range of frequencies, or not very frequent and have a wide range of sizes. This is not a flaw of the graph theoretic framework but rather a consequence of the extraction cost function used in the subgraph extraction algorithm. One can see that this function, as defined in chapter 8, is proportional to the size-frequency product of a discovered object. As such, during the frequent star search process, frequent stars with high size-frequency products tend to be chosen as the extracted objects. This does not mean that for a given frequent star search (i.e. one iteration of the subgraph extraction process), that a frequent star with a frequency higher than r_{min} combined with a size greater than k_{min} does not occur, but rather that another frequent star has a higher size-frequency product for the current search, even though its frequency may be r_{min} or its size may be k_{min}.

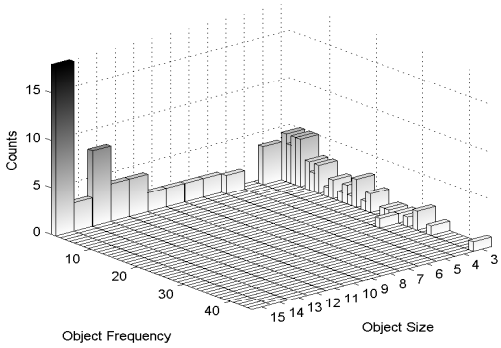
Another point to note with regards to the distributions of figure 9.8 is the high count for objects of size 15 in the case of all five audio excerpts. This is due to the upper limit placed on the object size in section 9.2.3. If this upper limit is increased or removed, bigger objects would be discovered and the peak in each histogram would



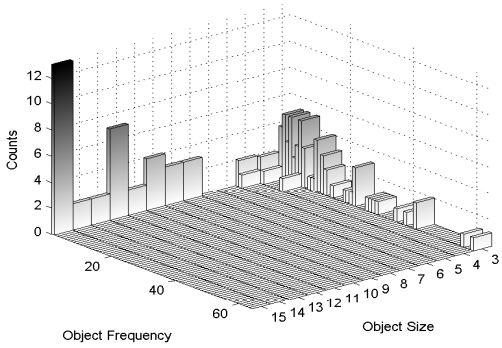
(a) Castanet



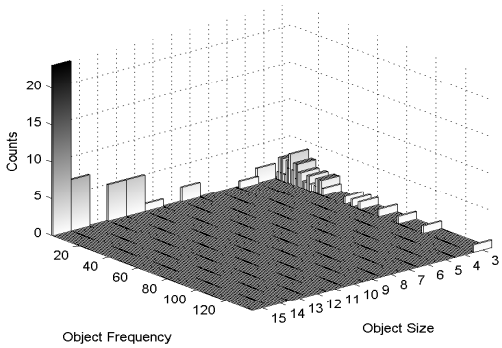
(b) Vibraphone



(c) Female Speech



(d) Male Speech



(e) Piano

Figure 9.8: Bivariate Distribution of first 100 Discovered Objects

be more dispersed. The removal of the pipeline mechanism used in the testbed would also lead to bigger objects and more frequent objects since all the vertices of all spikes in the spikegram could then be considered together. This would also lead to a smaller number of discovered objects than the numbers reported in table 9.5 since some objects could now be merged together. However, both the upper limit on the object size and the pipeline procedure are needed to ensure that the frequent star search process remains tractable, as mentioned in section 9.2. For this reason, the tractability of the frequent star search is one of the discussed topics in chapter 11 for future work.

Chapter 10

Conclusion

This thesis presents a novel graph theoretic framework for structure discovery. The contributions of the thesis at the application level are:

- the first time application of graph theory to the field of audio
- the discovery of new types of frequent audio objects, dubbed Non-Parametric Objects (NPO), which enable audio compression and differ from those currently found in existing parametric object coding schemes in that their shape is not restricted by any a priori psychoacoustic knowledge

At the theory level, the thesis contributes:

- a novel clustering algorithm when dealing with feature vectors such that distinct quality guarantees can be placed on each of the feature vector components
- a novel frequent subgraph mining algorithm such that frequent stars involving labelled edges can be mined in a single graph setting

Tests are performed using a corpus of audio files which spans a wide range of sounds. A testbed is constructed to evaluate the audio quality of the original spikegram and the reconstructed spikegram, both with respect to each other and with respect to the original audio file. The audio quality is evaluated using three metrics: the Signal to Noise Ratio (SNR), the Segmental Signal to Noise Ratio (SSNR), and the Perceptual Evaluation of Audio Quality (PEAQ) model. The overall and relative spikegram compression gains achieved by the framework are evaluated based on the extraction cost function designed for the experiment. Results show that the framework discovers new types of audio

objects which yield average respective overall and relative compression gains of 15.90% and 23.53% while maintaining an average audio quality of -0.395 on a PEAQ score scale.

Chapter 11

Future Work

11.1 Introduction

Although motivated by existing literature, the graph theoretic framework presented in this thesis is novel. As such, throughout its research and development, many unexplored subtopics of interest were encountered. The following sections discuss five main areas where further work would be promising and in some cases certainly improve the performance of the algorithm.

11.2 Tractability

One of the main concerns of the proposed framework is the tractability of the frequent star search in the subgraph extraction process of chapter 5. In this depth first search, every size-k frequent star that is deemed frequent can stem perhaps more than one frequent size-$k+1$ star. This leads to the generation of a very large number of frequent stars which can be seen as wasteful since, once all frequent stars have been generated, only one of them is selected and extracted.

The works in [50] and [51] propose a solution to this problem by pruning the number of generated subgraphs using heuristics. The main problem in both cases, however, is that the pruning is too aggressive and only small frequent subgraphs are generated. Often times this leads to both works completely missing out on generating the frequent star which would be selected as the optimal one. A secondary problem with these works is that the application of the heuristics requires such a heavy computational load that the overall execution time of these algorithms ends up being comparable to that of a

full frequent subgraph search such as the one in chapter 5. Although the authors of [52] claim to have solved the second issue, much work still remains to be done with regards to the first issue in the development of a tractable and efficient frequent subgraph search algorithm.

11.3 Other Subgraph Types and Isomorphism

As noted in chapter 3, the proposed framework only considers frequent stars as valid objects. This is because, in a star, a central vertex has a direct link (i.e. an edge) to all other vertices of the star. This allows this central vertex to be chosen as the anchor point when recording a frequent star as an object, as detailed in chapter 6. To allow for a greater possibility of objects, the door could be opened for other types of subgraphs to be considered as valid objects. For this to happen, the recording process of chapter 6 would need to be modified to allow the quality constraints imposed by the QT clustering algorithm to be respected even if not all the vertices in an object instance are directly connected to a central vertex (i.e. if not all the vertices are directly expressible as a function of a central vertex). It should be noted, however, that increasing the types of subgraphs to be considered as valid objects would also increase the execution time of the subgraph extraction process of chapter 5, where now, the frequent star search would no longer only be restricted to stars.

Along this same line, another issue for future work would be to allow frequent subgraphs to have instances which are isomorphic versions of each other. By definition, two subgraphs are isomorphic if there exists a one-to-one mapping between the vertices of one subgraph to the vertices of the other such that, in the case of this thesis, the edge labelling arrangement of the subgraphs is preserved. Any given subgraph only has a finite number of ways in which it can be transformed into an isomorphic version of itself. Each transformation can be indexed. The recording of a frequent subgraph as an object, as done in chapter 6, could then be easily modified when dealing with a frequent subgraph which has isomorphic instances by simply sending a transformation index for each instance in addition to the anchor points and the relationships of the object. Within the specific application of the framework to audio, the implications of isomorphic sounds from a psychoacoustic standpoint would also be a very interesting aspect to investigate.

The issue of isomorphism is very well studied in [53]. It can be checked for in two ways. The first approach is by verifying directly if a mapping exists between two subgraphs [53]. The problem with this method is that only two subgraphs can be checked at a

time. However, in [54] tips to increase the speed of this pairwise check are proposed. The second way of checking for isomorphism is by defining a fundamental label sequence (i.e. a canonical label) for each subgraph and checking whether two or more subgraphs have the same canonical label [53]. Even though the formation of a canonical label is time consuming since it requires that the edge labels be sorted in increasing order of label values, the advantage with this method is that more than two subgraphs can be checked for isomorphism at the same time. This method could also be easily integrated in the frequent star search of chapter 5 by using canonical labels instead of the instance identifiers currently used in the search. Either way, however, checking for isomorphism remains a computationally expensive operation [53].

11.4 Audio Object Parametrization

The objects found via the proposed extraction scheme are not subject to a predefined parameterization scheme. Although this allows for the discovery and extraction of a greater family of objects, the extracted objects can only be transmitted by way of a codebook. This requires that each extracted object be fully described once before other occurrences of the object can be reconstructed simply using the location of the occurrence and a reference to the fully described object in the codebook. As such, a compression gain can only be achieved if the object occurs more than once.

Further compression could be achieved if each extracted object could be efficiently parameterized using a minimum number of parameters. This would necessitate the discovery of mappings which would transform each extracted object to a common domain where they can be represented concisely.

Within the audio context, the discovery of such mappings would be great for the audio community as it would provide new and perhaps more informative and concise domains to represent audio signals. The a posteriori discovery of these parameterization spaces using the extracted NPO objects could then be used to develop new a priori defined object types in parametric object coding schemes.

11.5 Audio Object Versatility

One aspect of the discovered NPOs which remains to be investigated is how unique they are to the audio excerpt in which they are extracted. This could potentially become

an issue if the parametrization idea proposed in the previous section is to be investigated. If the objects are not versatile (i.e. cannot be found in different excerpts) their parametrization and subsequent usage in a parametric context would be moot unless the resulting parametric coder would only be used to encode the excerpt in question. One possible solution would be to filter out objects which are not common to more than one excerpt and only parametrize those which are versatile. However, much work remains to be done in this area.

11.6 Other Applications

This thesis presents the proposed audio object extraction framework in an audio coding context. However, the applicability to other audio and speech tasks remains up for investigation. One possibility would be to employ a slightly modified version of the framework in audio search tasks where one has already obtained a previously unseen audio object (i.e. an NPO) from an audio excerpt and wants to see if it exists in another excerpt. Other possible audio and speech applications include speech recognition and speech quality enhancement.

The framework is also most certainly applicable to tasks outside the realms of audio and speech. By using graph theory as its backbone, it makes the abstraction of any domain-specific representation possible using only vertices and edges. One such example would be vision where the representation in question is the neural response to an image. Another would be finance where stocks would then serve as a domain-specific representation of the market.

Appendix A

Notation

Notation used in the thesis presented in order of appearance:

- $\mathbf{x}$ – Audio signal waveform
- F_s – Sampling frequency
- N – Length of audio signal
- $\mathbf{g}$ – Kernel waveform
- h – Frequency channel
- $\mathbf{u}$ – Carrier signal waveform
- $\mathbf{w}$ – Window
- H – Set of frequency channels
- n – Sample index
- f – Normalized frequency
- c – Modulation index
- M – Window length
- M_{eff} – Effective window length
- p – Time instance

- X – Inner product coefficient
- N_{hop} – Hop size
- $\mathbf{s}$ – Spike
- s – Spike component
- S – Set of spikes
- G – Graph
- V – Set of vertices
- E – Set of edges
- v – Vertex
- e – Edge
- l – Label
- k – Size of star (number of vertices)
- $\mathbf{z}$ – Feature vector
- D – Number of dimensions
- Z – Set of feature vectors
- T – Threshold function
- C – Cost function
- k_{max} – Maximum size of star
- r_{min} – Minimum frequency of star
- r – Star frequency
- Q – Frequent star
- O – Set of object instances
- $\mathbf{o}$ – Object instance

- o – Object constituent
- ψ – Recording function
- δ – Representative relationship
- ζ – Reconstruction function
- N_{prune} – Pruning limit
- μ_{max} – Maximum allowable kernel displacement
- Γ – Number of amplitude quantization levels
- μ – Kernel displacement curve
- Λ – Overall cost
- N_{seg} – Length of segment
- σ – Upper bound
- k_{min} – Minimum size of star

Bibliography

[1] Information Technology. ISO/IEC 14496-3:2001, Information technology - coding of audiovisual objects - part 3: Audio. 2001.

[2] A. C. den Brinker, E. Schuijers, and W. Oomen. Parametric coding for high-quality audio. In *Audio Engineering Society Convention 112*, 2002.

[3] N. R. Reyes and P. V. Candeas. Adaptive signal modeling based on sparse approximations for scalable parametric audio coding. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):447–460, 2010.

[4] E. Vincent and M. D. Plumbley. Low bit-rate object coding of musical audio using bayesian harmonic models. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1273–1282, 2007.

[5] G. Cornuz, E. Ravelli, P. Leveau, and L. Daudet. Object coding of harmonic sounds using sparse and structured representations. In *Proceedings of the 10th International Conference on Digital Audio Effects*, pages 41–46, 2007.

[6] H. Purnhagen and N. Meine. HILN - the MPEG-4 parametric audio coding tools. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 3, pages 201–204, 2000.

[7] B. C. J. Moore. *An introduction to the psychology of hearing*. Elsevier, fifth edition, 2004.

[8] T. Irino and R. D. Patterson. A time-domain, level dependent auditory filter: the gammachirp. *Journal of the Acoustical Society of America*, 101(1):412–419, 1997.

[9] A. V. Oppenheim and R. W. Schafer. *Discrete-time signal processing*. Prentice Hall, third edition, 2010.

[10] P. I. M. Johannesma. The pre-response stimulus ensemble of neurons in the cochlear nucleus. *Symposium on Hearing Theory*, 1972.

[11] E. C. Smith and M. S. Lewicki. Efficient auditory coding. *Nature*, 439:978–982, 2006.

[12] P. Smaragdis. *Redundancy reduction for computational audition, a unifying approach*. PhD thesis, Massachusetts Institute of Technology, 2001.

[13] M. S. Lewicki. Efficient coding of natural sounds. *Nature Neuroscience*, 5:356–363, 2002.

[14] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural Computation*, 20(10):2526–2563, 2008.

[15] S. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

[16] C. Srinivasa. Pattern extraction for object-based audio coding - phase 1. Technical report, Communications Research Center of Canada, 2009.

[17] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems, and Computers*, pages 40–44, 1993.

[18] S. Krstulovic and R. Gribonval. MPTK: Matching pursuit made tractable. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 496–499, 2006.

[19] H. Najaf-Zadeh, R. Pichevar, H. Lahdili, and L. Thibault. Perceptual matching pursuit for audio coding. In *Audio Engineering Society Convention 124*, 2008.

[20] E. Vincent and M. D. Plumbley. Fast factorization-based inference for bayesian harmonic models. In *Proceedings of the 16th IEEE Signal Processing Society Workshop on Machine Learning for Signal Processing*, pages 117–122, 2006.

[21] A. T. Cemgil, H. J. Kappen, and D. Barber. A generative model for music transcription. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(2):679–694, 2006.

[22] H. Thornburg. *Detection and modelling of transient audio signals with prior information*. PhD thesis, Stanford University, 2005.

[23] L. Daudet. Sparse and structured decompositions of signals with the molecular matching pursuit. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1808–1816, 2006.

[24] P. Leveau, E. Vincent, G. Richard, and L. Daudet. Instrument-specific harmonic atoms for mid-level music representation. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(1):116–128, 2008.

[25] X. Rodet and F. Jaillet. Detection and modeling of fast attack transients. In *International Computer Music Conference*, pages 30–33, 2001.

[26] L. Daudet. A review on techniques for the extraction of transients in musical signals. In *Computer Music Modeling and Retrieval*, volume 3902 of *Lecture Notes in Computer Science*, pages 219–232. 2006.

[27] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[28] P. Smaragdis. Non-negative matrix factor deconvolution; extraction of multiple sound sources from monophonic inputs. In *Proceedings of the Independent Component Analysis and Blind Signal Separation Fifth International Conference*, volume 3195 of *Lecture Notes in Computer Science*, pages 494–499. 2004.

[29] M. Schmidt and M. Mrup. Nonnegative matrix factor 2-D deconvolution for blind single channel source separation. In *Independent Component Analysis and Blind Signal Separation*, volume 3889 of *Lecture Notes in Computer Science*, pages 700–707. 2006.

[30] M. Schmidt and M. Mrup. Sparse non-negative matrix factor 2-D deconvolution for automatic transcription of polyphonic music. Technical report, 2006.

[31] S. Wood. Non-negative matrix decomposition approaches to frequency domain analysis of music audio signals. Master's thesis, Universite de Montreal, 2009.

[32] J. C. Tilton. Image segmentation by iterative parallel region growing with applications to data compression and image analysis. In *Proceedings of the 2nd Symposium*

on the Frontiers of Frontiers of Massively Parallel Computation, pages 357–360, 1988.

[33] M. Belgin, G. Back, and C. Ribbens. A library for pattern-based sparse matrix vector multiply. *International Journal of Parallel Programming*, pages 1–26, 2010.

[34] J. Gao, Y. Hu, J. Liu, and R. Yang. Unsupervised learning of high-order structural semantics from images. In *IEEE 12th International Conference on Computer Vision*, pages 2122–2129, 2009.

[35] C. Cotton and D. P. W. Ellis. Finding similar acoustic events using matching pursuit and locality-sensitive hashing. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 125–128, 2009.

[36] R. Pichevar and H. Najaf-Zadeh. Pattern extraction in sparse representations with application to audio coding. In *European Signal Processing Conference*, 2009.

[37] T. Jehan. Perceptual segment clustering for music description and time-axis redundancy cancellation. In *International Symposium on Music Information Retrieval*, 2004.

[38] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. Wiley-Interscience, second edition, 2000.

[39] J. A. Hartigan. *Clustering algorithms*. Wiley, 1975.

[40] D. Pelleg and A. W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *International Conference on Machine Learning*, pages 727–734, 2000.

[41] L. J. Heyer, S. Kruglyak, and S. Yooseph. Exploring expression data: identification and analysis of coexpressed genes. *Genome Research*, 9(11):1106–1115, 1999.

[42] M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1038–1051, 2004.

[43] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the IEEE International Conference on Data Mining*, pages 721–724, 2002.

[44] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 11(3):243–271, 2005.

[45] J. Wang, Z. Zeng, and L. Zhou. CLAN: An algorithm for mining closed cliques from large dense graph databases. In *Proceedings of the 22nd International Conference on Data Engineering*, page 73, 2006.

[46] Y. Podolyan and G. Karypis. Common pharmacophore identification using frequent clique detection algorithm. *Journal of Chemical Information and Modeling*, 49(1):13–21, 2009.

[47] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Math*, 120(1-3):197–207, 2002.

[48] R. Pichevar, H. Najaf-Zadeh, and L. Thibault. A biologically-inspired low-bit-rate universal audio coder. In *Audio Engineering Society Convention 122*, 2007.

[49] R. Goldberg and L. Riek. *A practical handbook of speech coders*. CRC Press, 2000.

[50] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1(1):231–255, 1994.

[51] K. Yoshida and H. Motoda. CLIP: Concept learning from inference patterns. *Artificial Intelligence*, 75(1):63–92, 1995.

[52] M. Kuramochi and G. Karypis. GREW - A scalable frequent subgraph discovery algorithm. In *Fourth IEEE International Conference on Data Mining*, pages 439–442, 2004.

[53] S. Fortin. The graph isomorphism problem. Technical report, Department Of Computing Science, University of Alberta, 1996.

[54] B. D. MacKay. Practical graph isomorphism. *Congressus Numerantum*, 30:45–87, 1981.