Content-Based Retrieval of Digital Music

Thomas Sødring, B. Sc.

A dissertation presented for the degree of
Doctor of Philosophy

Information Management Group
School of Computer Applications
Dublin City University
Glasnevin

July 2002

Dedicated to

Nessa

Content Based Retrieval of Digital Music

Thomas Sødring, B. Sc

Submitted for the degree of Doctor of Philosophy

July 2002

Abstract

One of the advantages of having information in digital form is that it lends itself readily to content-based access This applies to information stored in any media, though content searching through information stored in a structured database or as text is more developed than content searching through information stored in other media such as music In practice, the most common way to index and provide retrieval on digital music is to use its metadata such as title, performer, etc , as has been done in Napster

My research has lead to the development of a digital music information retrieval system called CEOLAIRE which can index monophonic music files Music files are analysed for notes on the equal tempering scale, where note changes are observed and recorded as being up (U), down (D) or the same (S) relative to the previous note These note changes are then indexed in a search engine At query time, notes are generated by a user using a web based interface These notes form the query for the retrieval engine A user is presented with a ranked list of highly scored documents This thesis explores the building and evaluation of the CEOLAIRE system

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed : _____

ID No. : 98PD143

Date : 22/09/2002

Acknowledgements

I believe that the acknowledgements section of a thesis gives the reader a reflection of the contributions of others that go into a thesis These contributions should never be understimated In a sense, this thesis exists because of them, their support and loyalty over the years make this journey easier

The first people are the 'old gang', Jerh, Gerard, Jer, Dai, Rachael, Owen, Tony A special thanks to Tony for his advice and patience in helping me understand the DSP side of things The new gang, Kieran, Paul, Jiamin, Aidan, Wu-hai, Mary, Nano, Michelle, Mairead, Aoife Thanks guys, for the fun and being there during my more stressful moments I especially want to thank Mary, Nano, Aoife, Kieran and Hyo-won who read my thesis for me Thanks to Cathal, Hyo-won and Kieran for their critiques, much appreciated guys Can't forget the people I have lived with, Shelly, Avril, Nit, Conor, Mac and Ingvild Thanks to Sandra for her comments on music theory and showing me how to read music

I sincerely thank my supervisor Prof Alan Smeaton for all his support and guidance

I want thank Nessa for all her support and her never ending faith that I would one day finish this dissertation I also want to thank my parents for their ever present love and support

I would also like to thank David Bainbridge of the University of Waikato for allowing me the use of the NZDL MIDI folk-song collection and IBM for their generous donation of an IBM S/390 machine which provided many hours of exploration and enjoyment

I would also like to thank the organising committee of the 24^{th} BCS-ECIR European Colloquium on IR Research for funding my trip to Glasgow in March of this

year to present a paper on some of my current research

I gratefully acknowledge the School of Computer Applications for funding this work and Enterprise Ireland who funded the last few months of my Ph D through RIF grant # IF/2001/013

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Information Retrieval

1.1 Introduction

To date, human intellectual progression has posed many challenges and opportunities. One of the most important inventions of all time was the development of printing press by Johann Gutenberg in 1436. No longer was information the intellectual property of an elite few in society. A new era emerged, bringing with it a society where information was becoming accessible to everybody, even globally. Later, Martin Luther translated and disseminated the basic principles of the bible to lay people, to give them the opportunity to read and interpret information which previously had been the exclusive domain of the church hierarchy. Luther's stance is an example of how disseminating information gives power to people by giving people the ability to discuss and develop their own ideas rather than relying on a centralised source interpreting it for them. With the ability to disseminate, information is no longer centrally controlled, rather it is distributed and once information becomes distributed, the need to manage it effectively also becomes apparent.

After the World Wars, Western society experienced a massive industrial expansion. Europe was rebuilding itself after a destructive war, while the United States, which was relatively unscathed by the war on its territory, embarked on a research frenzy. This led to the generation of vast quantities of scientific literature. At that time, retrieval of information could only be achieved manually. As the quantity of information grew, so did the need to manage it. People were growing frustrated with

the inability to manage information effectively The computer revolution brought with it an ability to automate the process of information retrieval Thus a new field of research emerged and information retrieval was born

Microfiche was invented by Carl Carlson in 1961 However microfiche is only a method for browsing information, as it is a representation that is not directly suitable for information retrieval The development of the WWW was the next major advancement in the history of information dissemination, making information more accessible to more and more people, potentially reaching every person in the world The important tool for web retrieval is the use of search engines which try to create an index for the web However, the nature of web data makes this a difficult task A lot of traditional text search engine tasks have been explored and clever solutions have been developed and adapted, including issues of scale as search engines are now able to index billions of documents Today, information retrieval covers more than just access to text Now it includes multimedia, audio and image and even video which integrates image, audio and text The problem which we are concerned with in this thesis, is that of automatic information retrieval of audio and within that, specifically music, rather than speech Figure 1 1 shows a taxonomy of the different media upon which we can perform information retrieval

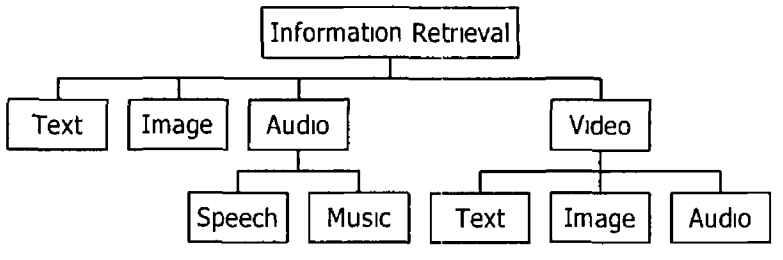


Figure 1 1 Different information retrieval media

1.2 Information Retrieval

Information retrieval can be undertaken on information extracted from sources which can be structured, unstructured or semi-structured An example of a structured source of information is a relational database A relational database structures information into tables Within tables, data is stored in attributes (columns) The

unit of information is normally a tuple (a row) or part of a tuple of data from a database. Querying information from a structured database of information is achieved using a query language (such as a structured query language or SQL). An example SQL query is shown in Figure 1.2. Retrieving information from a database is a relatively straightforward task. As data is stored in cells as rows and columns, the location of every piece of related data is known exactly.

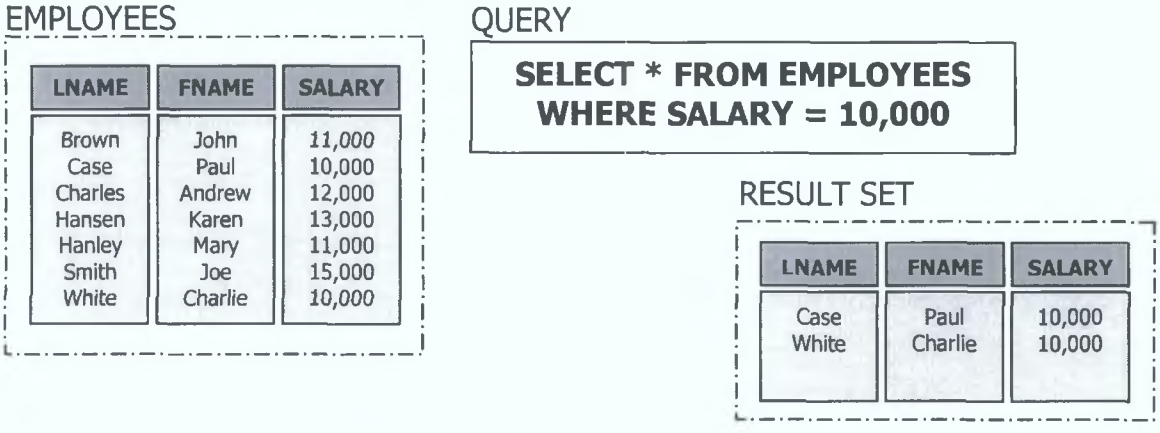


Figure 1.2: An example SQL query

Performing information retrieval on text, such as text taken from the WWW, is more problematic. The information resides in documents so we are not concerned with information retrieval, rather we are concerned with document retrieval. Documents from the web are semi-structured. The content of a web document is marked-up using a hypertext mark-up language, HTML, which infers a structure on a document. [Baeza-Yates et al., 99b, p369] identifies a number of challenges with data on the web. These include:

- **Distributed data**. Web data sits on many different types of computers and platforms, without any predefined topology. Bandwidth and network reliability to allow access to WWW data is not guaranteed.

- **Volatile data**. Web servers are constantly being added and removed to and from the web. The same applies to the content of web documents, many of which are in a constant flux, with their content being updated, deleted and augmented. Content can be relocated and documents can be moved or renamed.

- **Large volume**. Since its inception, the WWW has consistently undergone growth in content on an exponential scale.

- **Unstructured** and **redundant data**. A HTML document is not well structured. That is, the content of the document is marked up using mostly formatting tags, rather than tags which describe exactly their structure and content.

- **Quality of data**. Due to the lack of a central editorial process for information on the web, data can be false, invalid, poorly written and can easily contain grammatical errors.

- **Heterogeneity of the data**. The WWW is global, therefore it draws from a multi-lingual and multi-alphabet domain.

These challenges illustrate that building a process to allow for the searching of WWW text data is by no means trivial.

Unstructured information sources are sources of information which have not had their content marked up using a mark-up language. Raw multimedia data is unstructured. By this we mean the content has not been marked up. In fact the data may contain structure but this structure has to be observed (automatically or manually). In recent years we have seen research into how we can structure this type of information so that it can be processed effectively. For example, dealing with news-wire text, raw audio [Smeaton et al., 98] and video [Smeaton et al., 01]. Although analysis and conversion of these data sources is possible, historically it has been a manual process, as in the case for video where humans edit and tag content-bearing information.

In the work reported in this thesis, as we wish to perform information retrieval on music, we can ask whether music is structured or unstructured? The answer depends on which format we use when storing music. Formats like MP3, WAV etc. contain no mark-up tags from which we can extract information relating to content. A format like MIDI has some structural information which can be used when attempting retrieval. These include properties like tempo and which instrument should be utilised when playing notes. The amount of music on the web does grow, but not to the same extent that HTML web documents do. The popularity of Napster brought

an enormous amount of music to the Internet, but the Internet as a channel for music distribution has not yet proved commercially viable, except perhaps for the case of radio stations broadcasting Internet music streams over TCP/IP. Neither are music documents subject to the same flux of change in terms of content: once a piece of music has been composed and stored somewhere - whether on the web or in a database or a repository - it is rarely modified or updated. This stability that music documents have removes at least one of the complexities that text information retrieval on the web has to overcome.

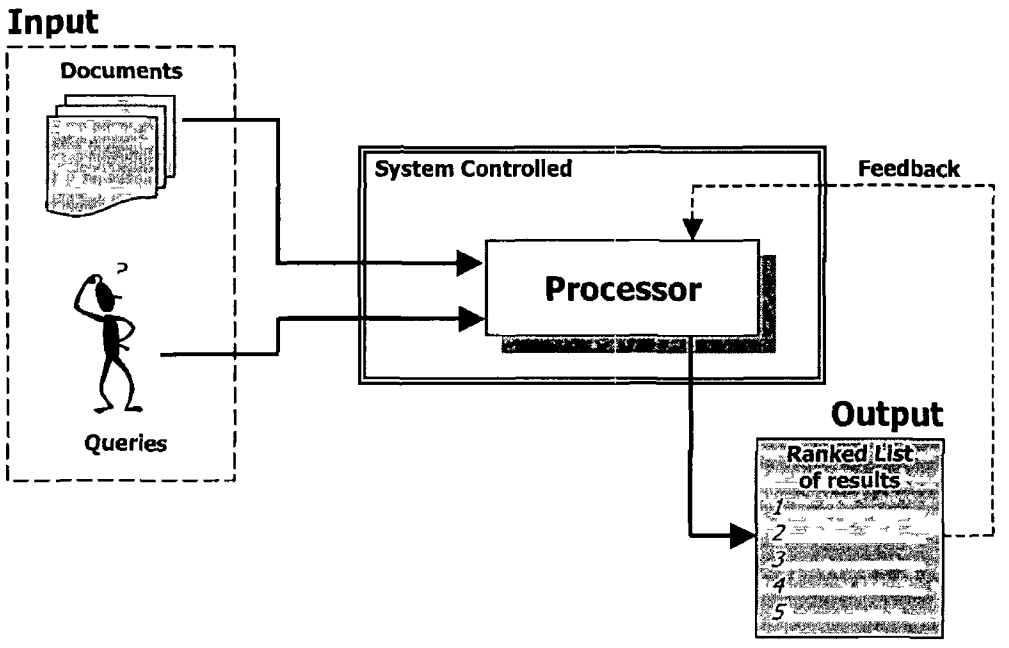
1.3 Principle Components of an IR System

The process of automatic information retrieval can be broken into four important components: *input*, *processing*, *output* and *relevance feedback*. The input component comprises of documents and user queries. The processing component is facilitated by documents (indexing) and queries (matching queries against documents). The output component consists of a ranked list of documents which the processing component deems relevant to the user query. The relevance feedback component takes relevance judgements by a user to further expand or limit the answer set. The four principle components of an information retrieval system are shown in Figure 1.3.

If we take a closer look at this model, we can break the components down into 4 distinguishable functions, *gathering*, *indexing*, *searching* and *management* [Agosti et al., 00]. Documents first have to be retrieved to generate a corpus. Once a corpus is available the documents are indexed, so that they can be searched. With a system in place, it has to be managed effectively to ensure that the index is always up to date and correct if the document collection or corpus is anything but unchanging.

1.3.1 Gathering

Documents can be collected automatically or manually. Manual gathering implies that a collection of documents is available somewhere and is static. Automatic document gathering is usually performed on a non-static collection e.g. the WWW, by a spider or a crawler which crawls the web seeking out and downloading new



together

Figure 1 3 Principle components of an information retrieval system

documents Once documents are downloaded, some pre-indexing tasks can be undertaken, such as irrelevant mark-up removal, stop-word removal and stemming The collection we used as part of our researched was manually gathered and is static

1 3 2 Indexing

When the corpus of documents which have been gathered is ready to be indexed, documents are processed to generate an index A variety of data structures and programming languages can be used to achieve this Typically, compiled languages (C, C++) rather than interpreted languages (Java, PERL) are used to implement these processes This is for speed and memory storage requirements, as programs written using compiled languages usually run faster and have lower memory requirements than those written using interpreted languages

1 3 3 Searching

The searching component of an information retrieval system takes a user's information need expressed as a query and applies some kind of relevance estimate (Boolean, vector space or probabilistic similarity) against the corpus This process usually re-

sults in a list of documents (answer set) deemed relevant by the system If the relevance estimate is Boolean, then no meaning can be inferred from the ranking in the answer set The probabilistic model and vector space models both return a ranked list of documents, with the high scoring documents ranked first Relevance feedback is also part of the searching component whereby judgements made by a user about the relevance of documents can be used to automatically modify a query The underlying assumption behind most relevance feedback theories is based around the occurrence of terms in documents that a user deems relevant The terms which occur more frequently in relevant documents than in non-relevant documents can be used to help retrieve other relevant documents Relevance feedback could play an important within a music information retrieval system However, within the scope of this dissertation, relevance feedback plays no role

1 3 4 Management

It is not enough to simply gather a corpus and index it for subsequent retrieval The source for a corpus may naturally change over time, so the gathering and indexing component will have to be re-run frequently In the case of the WWW, we have discussed the nature of the data and we can also observe that links between web documents die sporadically, connections fall, documents move, document content changes, etc An information retrieval system has to try and overcome all of this

1.4 Indexing

As we have already seen, one of the central components of an information retrieval system is to create an index for a corpus of documents, where a user can consult the index in the form of a query to locate the occurrence of a term in a document within the corpus Important factors when deciding the data structure for an index are speed and storage requirements There are a number of ways in which we can index documents

1 4 1 Original Format

The simplest way we could implement a text based information retrieval system is to store the documents as text files and every time a query is issued, a string comparison could be undertaken to match every word in every document against every word in the query While this naive approach could be deemed an information retrieval system, it would be unsustainable for large collections of documents, as it could take days to resolve a query

More efficient methods have been developed, including using intermediary data structures to store the data, which speeds up the process of resolving a query One such data structure is an inverted file

1 4 2 Inverted File

An **inverted file** [Berry et al , 99, pp 27-29] is a data structure which allows information retrieval systems to track which documents contain a particular term within a corpus Traditionally, terms belong to documents but with an inverted file, documents belong to terms For every term in the index, an associated list of documents, which contain that particular term, is recorded This index is an alphabetically sorted data structure from "a" to "z" Every time a new term is encountered, its index location is calculated and a document identifier is added to the list of identifiers associated with that term Inverted files are very powerful and essential features of an information retrieval system, as they allow for the swift look-up at query time, of the list of documents associated with a particular term An inverted file can have weights associated with each document identifier which can be used by the query manager when deciding how to rank documents in order of relevance to a query An inverted file is illustrated in Figure 1 4 The main disadvantages with inverted files are that updates to the index are slow They also have expensive storage requirements The storage requirements for an inverted file can be up to half that of the original text [Witten et al , 99, pp 114-115] Another approach to indexing documents which can result in a storage requirement as low as 10% of the original text corpus is the use of a signature file

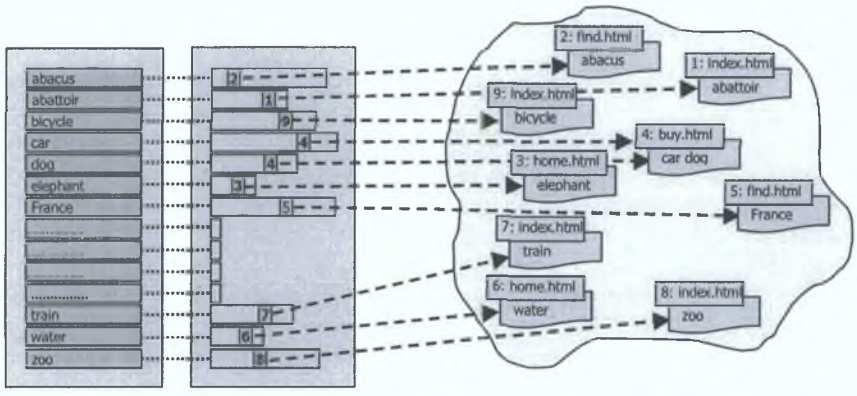


Figure 1.4: Inverted file

1.4.3 Signature Files

Signature files [Berry et al., 99, pp 27-29] are another approach which allow a corpus of documents to be converted into an intermediate representation for subsequent search. Central to the use of signature files are hash signatures or hash values. Signature files work by breaking the input text into fixed length blocks and for each term in each block, applying a hash function to generate a hash value for that term. The hash values (which is a binary number) for each term are OR'ed to produce a signature for the block. Figure 1.5 shows how binary signatures are generated after the hash function is applied on a block of terms.

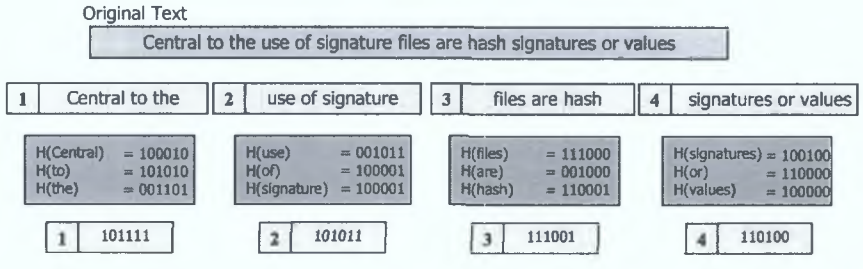


Figure 1.5: Signature files

At query time, a query term is processed by the same hash function generating a binary hash-number. Signature file hashes and the query hash are compared sequentially using the bit-wise AND operator, allowing for exact or partial matching. If the digits are the same, any bit-string hash value when AND'ed with another bit-string hash-value will result in the generation of the same number. Figure 1.6 shows a query being matched and the occurrence of a false-drop (when an incorrect match is found). If the result is the same as the query term's number, then it is assumed to

be a potential match A potential match has to be confirmed as a match to ensure it is not a false-drop and the block is checked to ensure that the term is present A false-drop occurs on the first comparison but it can be resolved by checking the hash values of the other terms in the block

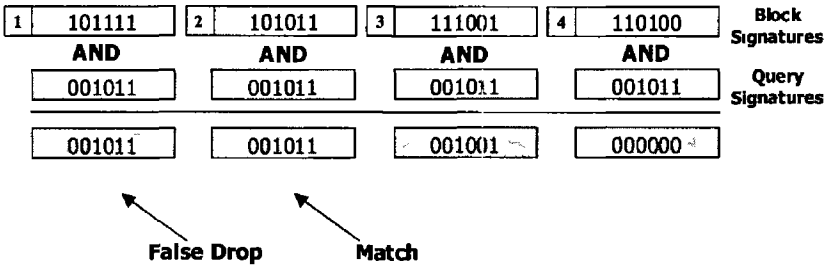


Figure 1 6 Term matching in a signature file

A limitation of signature files is that they do not have the potential to scale to the same extent as inverted files as the search through the file is sequential As the file grows, the search time also grows, but they are more efficient for phrase searching as the spatial location of terms is kept intact We will not use signature files within this dissertation, rather we have included an explanation so the reader is aware of alternative methods which retrieval can be undertaken

1.5 Retrieval

The task of retrieval is as follows given a number of documents, how can we select documents that best match the user's information need expressed as a query Currently we select documents by scoring them using corpus and query weights Three popular approaches to implementing the retrieval component of an information retrieval system are the *Boolean model*, *vector space model* and the *probabilistic model*

1 5 1 Boolean Model

Boolean model information retrieval systems are simple retrieval systems based on set theory and Boolean algebra [Baeza-Yates et al , 99b, pp 25-27] The Boolean model formulates a query as a combination of terms, conjugated using the Boolean

operators **AND**, **OR** and **NOT** Finding documents in a corpus about "Audio Retrieval" which do not contain the word "Speech", would result in a Boolean query "(**audio** AND **retrieval**) AND NOT **speech**", matching indexed terms based on whether they are present in a document or not Figure 1 7 shows a set theory representation of the Boolean expression "(**audio** AND **retrieval**) AND NOT **speech**"

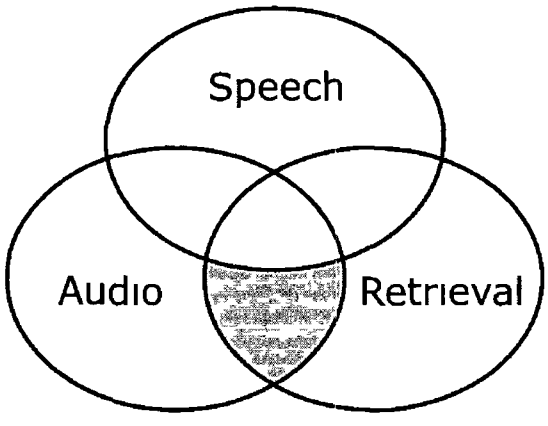


Figure 1 7 Boolean set theory

A feature of the Boolean model is that an answer set can easily contain too many, or too few results A document which may not match a Boolean query 100%, may still be relevant A substantial improvement in retrieval can be achieved if an information retrieval system were able to partially match queries to documents One method to achieve partial matching is to use the *vector space model*

1 5 2 Vector Space Model

The vector space model [Salton et al , 75] can partially match queries against documents This is achieved by implementing a non-binary term weighting scheme to estimate the degree of similarity between a query and the documents indexed by the processor The concept is shown graphically in Figure 1 8 where we can see that document 1 has a higher similarity to Q than document 2 This can observed by the fact that the angle between Q and d_1 is smaller than the angle between Q and d_2 and thus the d_1 is deemed closer the query Q Allowing for degrees of similarity between documents and a query, allows the processor to rank the output in terms of higher scoring documents

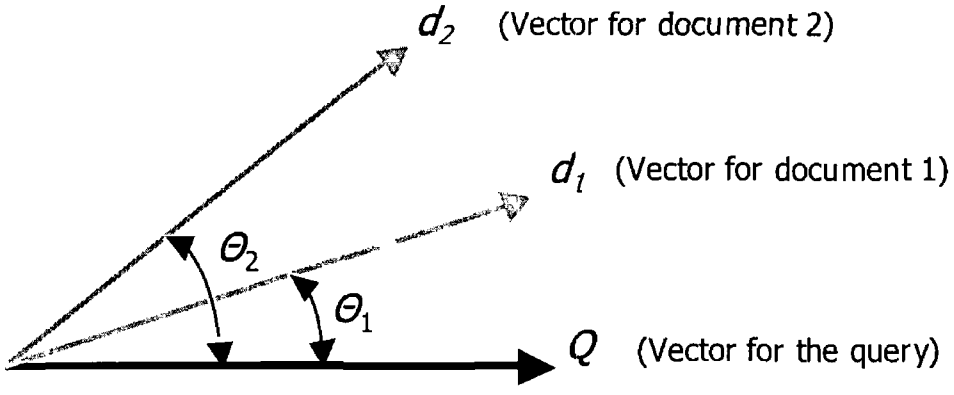


Figure 1 8 Vector space model showing similarity between two documents and a query

One of the more popular methods for generating the weights behind the vector space model makes use of *tf*idf* weighting *tf*idf* weighting revolves around the idea that the more often a term appears in a document (the higher its term frequency or *tf*), the more that term describes the content of the document It also takes into account how often that term appears in all documents (the inverse document frequency or *idf*) The simple formula for *tf*idf* weighting is

$$w_{ij} = tf_{ij} \left(\frac{N}{df_j}\right)$$

where

w_{ij} = weight of term j in document i

tf_{ij} = occurrence count of term j in document i

N is the number of documents in the corpus

df_j = the number of documents in the corpus which contain term j

The idea behind *tf*idf* weighting is straightforward It makes use of a number of important, yet easily computed factors The first of these is the number of times a particular term appears in a given document (tf_{ij}), the second is the number of documents that contain this term at least once (df_j) and finally the total number of documents (N) The basic idea is, that a term which appears across many (if not all) documents, is not a good document discriminator and should be ranked low A

term which occurs often in a few documents, should however be ranked higher By assigning N=2000, tf_{ij} =20 and df_j =10, then $w_{ij} = 20\ \left(\frac{2000}{10}\right) = 400$ Here term i occurs in a few documents (10 in total) so it gets a high weighting Now, if we assign N=2000, tf_{ij} =20 and df_j =1900, then $w_{kl} = 20\ \left(\frac{2000}{1900}\right) = 21\ 1$ It should be clear that the more frequent a term appears across all documents, the lower its weighting Query weights are generated using the above formula and retrieval is achieved by scoring the query term weights by document term weights This is implemented by multiplying the query term weights against the appropriate document term weights, resulting in a list of documents which can be ranked in decreasing order by scores

Other methods exist for generating information retrieval scoring weights One such model attempts to define the retrieval paradigm within a probabilistic framework, known as the *Probabilistic model*

1 5 3 Probabilistic Model

The Probabilistic model [Robertson et al , 76] is an alternative model which can be used for information retrieval Scoring weights are generated for documents by attempting to predict the probability that a given document will be relevant for a particular query Documents are then ranked in decreasing order of their probability of relevance, through the use of decreasing scores The probabilistic model assumes binary relevance and mutual independence, that is, a document is either relevant or not relevant to a query The relevance of one document has no bearing on the relevance of another As with the vector space model, term frequency and document frequency are used as indicators of relevance A common implementation of the probabilistic model is BM25 BM25 uses different formulae for generating weights for query terms and index terms The formula used for generating weights at **index time** is

$$w_{ij} = \frac{(k_1+1)tf_{ij}}{K+tf_{ij}} \quad \text{where} \quad K = k_1\left[(1-b) + b \bullet \frac{l_i}{advl}\right]$$

where

tf_{ij} is the frequency of term j in document i

w_{ij} is the weight for term j in document i

b and $k_1$are parameters

l_i is a count of the number of terms in the document

K is a figure which represents the ratio between the length of document i and the mean collection document length ($advl$)

$advl$ is the mean collection document length which is the average number of terms per document

The formula used for generating weights at **query time** is:

$$w_{qj} = \frac{tf_{qk}}{k_3+tf_{qk}} \cdot ln\left[\frac{(N-df_j)}{df_j}\right]$$

where

w_{qj} is the weight for term j in the query, q

tf_{qk} is the frequency of term k in the query q

k_3 is a parameter

N is the number of documents in the corpus

df_j is the number of documents which contain the term j

There are quite a few other models for information retrieval and the area of developing these models is quite an active topic. For example, Bayesian networks, logical models and language modelling are all hot topics in information retrieval research, but these are beyond the scope of this discussion.

1.6 Evaluation

An important step in the process of building an information retrieval system is to evaluate its performance with respect to the quality of the answer set. The answer set is the list of documents ranked in terms of weighting scores, that the processor deems relevant to a users query. Two different indicators can be used to evaluate retrieval performance, *precision* and *recall* [van Rijsbergen, 79, pp 114-115]. Precision and

recall are indicators of the quality of retrieval at a given point in the answer set.

1.6.1 Precision

Precision is used to give an indication of the relevance of the answer set. It is the number of relevant documents in the answer set, divided by the total number of documents in the answer set. See Figure 1.9 for a graphical representation of precision. In this figure we are observing two shades. The darker shaded portion of the figure represents all the documents within the answer set while the lighter shaded portion represents the documents in the answer set which are relevant.

$$Precision = \frac{\#\ of\ relevant\ documents\ in\ the\ answer\ set}{\#\ of\ documents\ in\ the\ answer\ set}$$

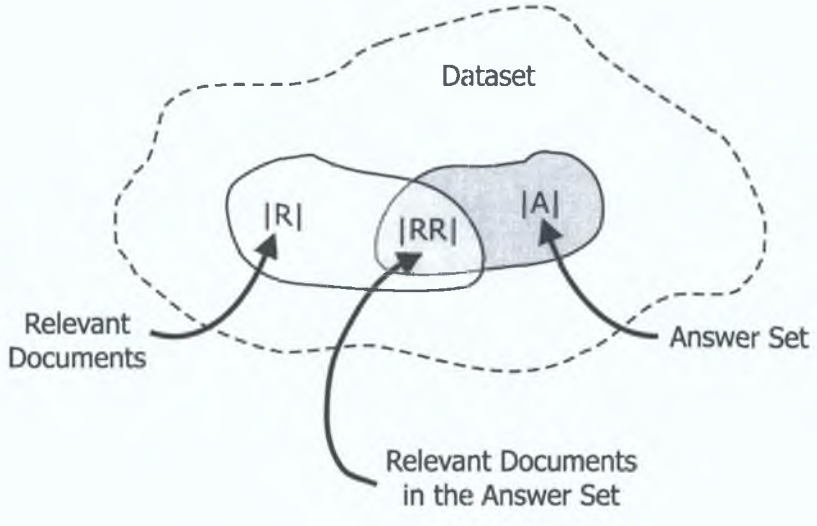


Figure 1.9: Demonstrating precision

1.6.2 Recall

Recall gives an indication of how many of the documents returned are correct. It is defined as the number of documents that the processor deems relevant (number of documents in the answer set), divided by the number of documents that are relevant within the corpus. See Figure 1.10 for a graphical representation of recall. The darker shaded portion of the figure represents all the documents within the corpus which are

relevant while the lighter shaded portion represents the documents in the answer set which are relevant.

$$Recall = \frac{\#\ of\ relevant\ documents\ in\ the\ answer\ set}{\#\ of\ relevant\ documents\ in\ the\ corpus}$$

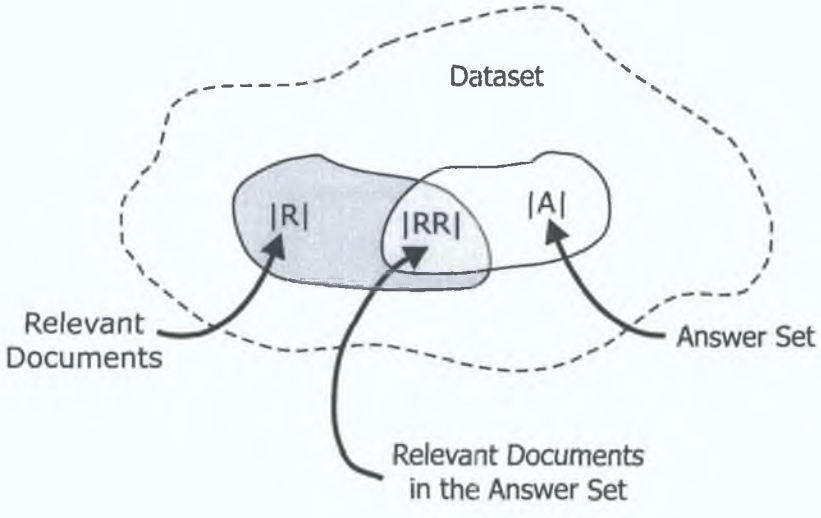


Figure 1.10: Demonstrating recall

1.6.3 Plotting Precision Versus Recall

If the answer set is ranked with the highest scoring document first, precision and recall values will vary as the user steps through the answer set. These values can be plotted against each other, resulting in a graph showing the strength of the answer set. The upper bound for both precision and recall is 1. With regard to precision, a value of 1 means that all of the documents in the answer set are relevant and with recall it means that all of the relevant documents in the corpus have been retrieved.

Consider a case where an information retrieval system retrieves 12 documents in response to a particular query, and a specialist has deemed 10 documents within the corpus as relevant. If the first document in the answer set is one of the relevant ones, then at that point there is a precision of 1 and a recall of .1 (10% of all relevant the documents are retrieved). If the second document is relevant, then the system has a precision value of 1 and .2 recall. If the third document returned by system is not marked as relevant, then the system has .66 precision and .2 recall (2 out of 3

retrieved documents are relevant and 2 out of 10 relevant documents are retrieved) A typical precision versus recall graph averaged over a set of queries (50 for example) is shown in Figure 1 11

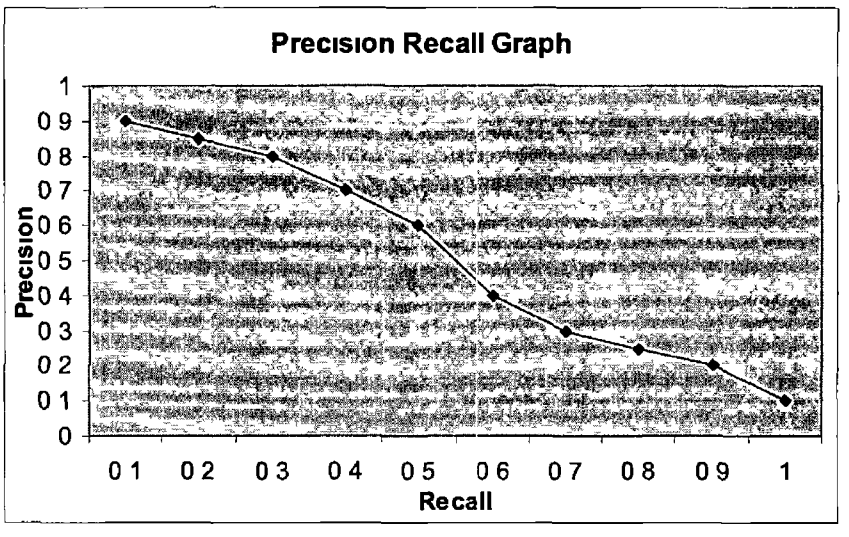


Figure 1 11 Precision versus Recall

When evaluating an information retrieval system, precision versus recall figures are usually generated for each query submitted to the system and these are then averaged to compute an average set of precision and recall values Given the fact that a user often only views the top 10 ranked documents resulting from a search then averaging the precision figure at 10 documents could be a useful measure of retrieval performance, i e how many documents ranked in the top 10 results are actually relevant This is often carried out at certain cut-off values such as 5, 10, 15, 20, 30, 100, 200, 500 and 1000 documents

A limitation of using recall is that knowledge of all documents relevance to queries in the collection is assumed in advance However, it simply may not be possible to discover this knowledge for large collections

1.7 Goal of Thesis

The goal of this dissertation is to explore how we can build and evaluate an information retrieval system for music We are concerned with Western music only, that is, music with 12 notes between each octave, although the principles we will

apply can be adapted to other musical scales We are dealing with music as an unstructured medium from which we will automatically extract note information As we will discover later, musical melody can be represented using strings With this in mind, we can thus index music using traditional information retrieval techniques for subsequent retrieval We have built a music information retrieval system called CEOLAIRE which can index music taken from raw audio files for subsequent retrieval Throughout this dissertation, we present the reader with the CEOLAIRE system which we have used as a vehicle for experimenting with various aspects of music information retrieval We aim to show through our performance evaluations of CEOLAIRE, a framework for building and evaluating a musical information retrieval system

1.8 Summary

This chapter has set the scene for the rest of this dissertation We have defined what information retrieval is about and the components required to build a basic information retrieval system We have reviewed some of the more common approaches applied to both indexing and retrieval Finally, we discussed evaluation of an information retrieval system

The rest of this thesis is organised as follows Chapter 2 presents the reader with an introduction to audio and audio processing we will look at how audio which is a continuous signal can be recorded in order to be manipulated In Chapter 3 we take an in-depth look into how we can gain access to musical content from audio files Chapter 4 presents a literature review of some of the different music information retrieval systems, both past and present In Chapter 5 we introduce the CEOLAIRE music information retrieval system which we developed and upon which this dissertation is based Chapter 6 and 7 are concerned with the evaluation of CEOLAIRE, Chapter 6 evaluates CEOLAIRE'S extraction engine while Chapter 7 evaluates CEOLAIRE'S retrieval performance

Chapter 2

Audio and Audio Processing

2.1 Introduction

This chapter presents an introduction to audio and some of the properties associated with audio. Later on we explore how audio can be stored, manipulated and processed digitally, but first we will present some basic principles relating to the physics of sound.

Sound exists as pressure waves travelling through a medium e.g. air or water. It is a continuous wave consisting of compressions and rarefactions of neighbouring particles of the medium through which it is travelling. In the upper part of Figure 2.1 we can see the compressions and rarefactions of the neighbouring particles of the medium that sound travels through. In the lower part of the same figure we can see what this look likes when plotted as a graph. In between compressions and rarefactions, the lack of dots show that there are neither compressions nor rarefactions.

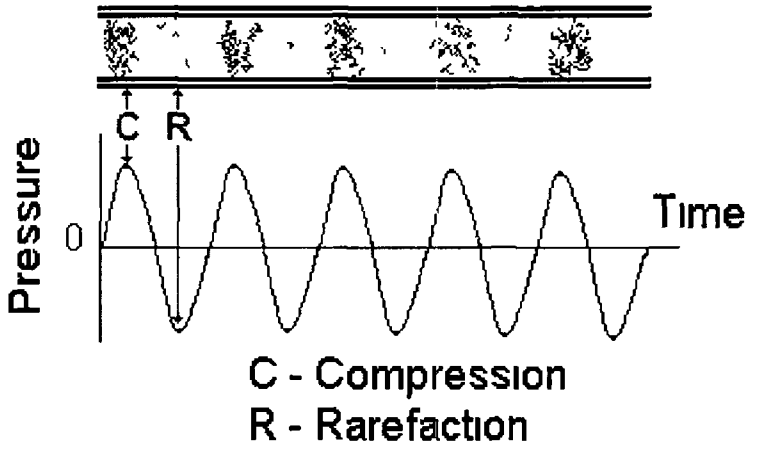


Figure 2 1 Sound as a continuous wave of compressions and rarefactions

2.2 Properties of Sound

Sound is a pressure wave as shown above Figure 2 2 below shows a picture of a sine wave identifying the **crest**, the highest displacement of the wave from the medium, and the **trough**, the lowest

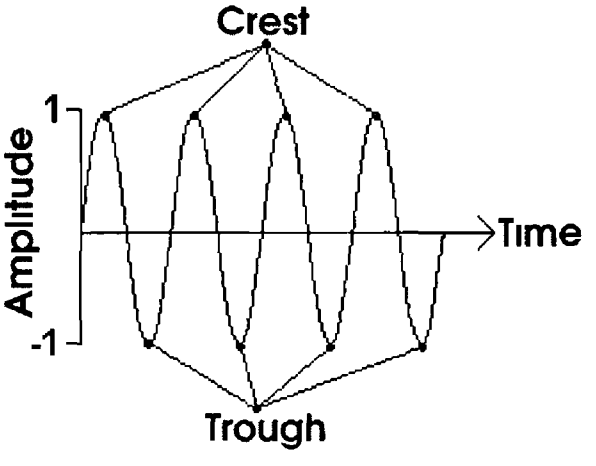


Figure 2 2 Crest and trough

2 2 1 Phase

A full rotation within a circle goes through 360° and a sine wave can also be measured in degrees A sine wave goes through a 360° **phase** shift before it starts to repeat Figure 2 3 illustrates some of the different phases of a sine wave until it repeats

Figure 2.3: Different phases of a sine wave

2.2.2 Frequency

The **frequency** of a wave is the number of times it repeats per second. The unit of measurement for frequency is Hertz (Hz). The frequency of the wave shown in Figure 2.4 is 4 Hz, this means that it repeats four times every second. A property which is related to frequency is *pitch*. The perceived pitch of a sound is the ear's response to frequency and can be observed in absolute terms whereby the pitch of a sound is recognised without any other sounds present or in relative terms where the pitch is recognised by the interval (difference) between two sounds. That is, if one sound has a higher frequency than another, it is said that it has a higher pitch. Conversely, if one sound has a lower frequency than another it has a lower pitch.

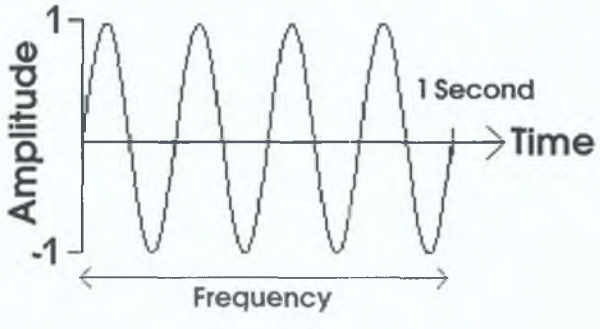


Figure 2.4: Frequency of a sine wave

2.2.3 Amplitude

The **amplitude** of a wave is the distance from the medium to the top of the crest or the bottom of the trough. It is the maximum displacement in either direction from the equilibrium position. In the case of simple sine waves, the distance is normally equal, but it can vary for more complex waves. Figure 2.5 shows amplitude. In the case of sound waves it represents loudness i.e. the louder a sound the greater the

maximum displacement

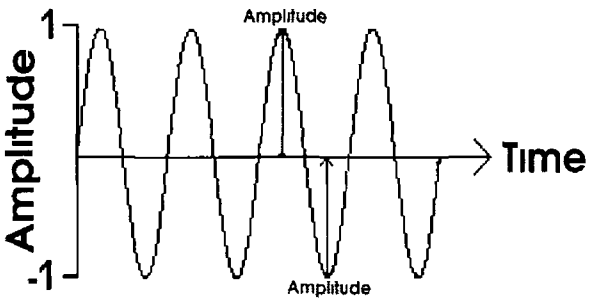


Figure 2 5 Amplitude of a wave

2 2 4 Wavelength

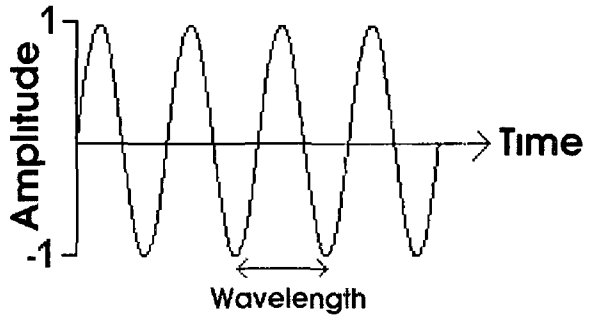


Figure 2 6 Wavelength

The **wavelength** of a wave is the distance from one crest or trough to the next

2 2 5 Period

The **period** of a wave is the time taken for the wave to complete one cycle, that is, the time taken for the wave to rotate through a 360° phase shift Period is directly related to frequency as its reciprocal, $period = \frac{1}{frequency}$ For example, a wave with a frequency of 4 Hz, that is, it repeats 4 times second, has period of 0 25 seconds It takes 0 25 seconds for the wave to go through a 360° phase shift

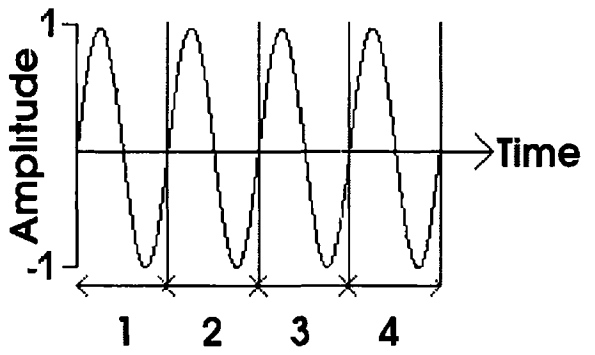


Figure 2 7 Period

The properties we have explored here lay the foundation for the understanding of the basic principles of audio needed when we later review methods for content based access to digital audio In particular, phase, frequency / pitch and amplitude

2.3 How We Hear Sound

Humans and animals have evolved a complex bio-mechanical mechanism to allow us to "hear" sounds The ears are used to channel continuous sound from the medium they are travelling through to the inner structures of the ear where they are converted into electrical impulses interpretable by the brain The ear can be divided into 3 distinct sections

- The Outer ear
- The Middle ear
- The Inner ear

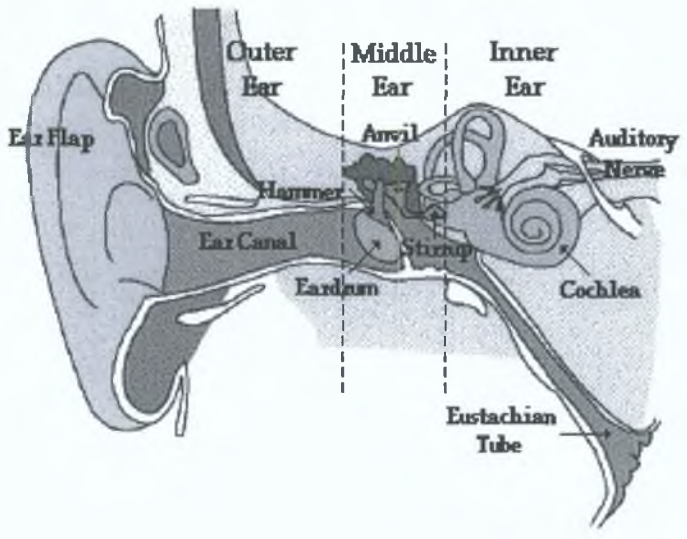


Figure 2.8: The human ear
Reproduced from [Glenbrook, 02]

The outer ear simply collects and channels sound to the middle ear, which in turn transforms the sound energy into the internal vibrations of the bone structure of the middle ear. These vibrations then create compressional waves within the fluid of the inner ear which are transformed into nerve impulses, which in turn are transmitted to the brain. The middle ear is an air filled cavity consisting of the eardrum and three tiny bones, namely

- The Hammer
- The Anvil
- The Stirrup

The eardrum is a tightly stretched membrane which vibrates when a sound wave exerts pressure on it. Remember a sound wave is made up of compression and rarefactions. A compression exerts a push inwards and a rarefaction exerts a pull outwards. This motion is picked up by the eardrum and sets the hammer, anvil and stirrup vibrating at the same frequency as the sound wave. The vibrations are then transferred to the fluid of the inner ear creating a compression within the fluid.

The inner ear consists of:

- The Cochlea

- The Semi-circular Canals
- The Auditory Nerve

The cochlea is a small snail-like organ containing about 20,000 hair-like nerves. The nerve cells differ in length by miniscule amounts and have different degrees of resiliency to the fluid which passes over them. Each hair-like nerve has a natural frequency and when a compression wave with a frequency that matches that natural frequency passes over it, it resonates with a larger amplitude of vibration. This then causes an electrical impulse to be sent on the auditory nerve to the brain. The semi-circular canals provide no help in hearing, rather they assist with balance and in detecting accelerated movements. The human ear is capable of detecting sounds in the range of 20 Hz to 20,000 Hz.

The difference between the intensities of the quietest sound that we can hear and the loudest non-damaging sound is on a linear ratio of $1 : 1.2*10^{12}$, although the ear is not actually capable of distinguishing between this many sound intensities. Rather, sound is perceived logarithmically, that is, the difference between two sounds is not simply the difference between the two intensities, but the ratio of the logs of the two intensities. For example, when comparing values from Table 2.1 for the intensity level of a street with no traffic with that of a normal conversation we can see that the two intensities differ by a factor of a thousand, yet we do not perceive them differing by this much. In fact, one sounds twice as loud as the other. What happens is that the perceived difference between the two sounds is the ratio of the logarithm of their relative intensities

$$log_{10}(1,000) : log_{10}(1,000,000)$$
$$3 : 6$$
$$1 : 2$$

Thus, sound intensity is measured in decibels (dB), based on a logarithmic scale. The lowest sound that we can hear, called the *threshold of hearing,* is assigned the value of 0 dB, with the loudest non-damaging sound assigned the value of 120 dB.

The average person is able to distinguish sound pressure level differences of about 3 dB.

Sound Level	Decibels	Intensity
Threshold of hearing	0 dB	1
Leaves rustling	10 dB	10
Someone whispering (1m)	20 dB	100
City street with no traffic	30 dB	10^3
Classroom, office	50 dB	10^5
Conversation	60 dB	10^6
Busy city street	70 dB	10^7
Jackhammer (1m)	90 dB	10^9
Pop concert	110 dB	10^{11}
Threshold of pain	120 dB	10^{12}
Jet engine (< 50m)	130 dB	10^{13}

Table 2.1: Decibel levels and their sound intensities
Intensities are relative to $10^{-12} Watts/meter^2$
Table taken from [Haliday et al., 97]

The ear is most sensitive to frequencies between 500 Hz and 4,000 Hz and tends to amplify them more than frequencies outside of that range. This band corresponds closely to the band of frequencies which governs speech. If two sounds of different frequencies but of the same amplitude were played, one at around 3,000 Hz and the other at 8,000 Hz, the 3,000 Hz signal would be perceived as being louder.

2.4 Sampling

Audio is a continuous signal and must therefore be sampled in order for it to be stored digitally. Sampling involves measuring and recording pressure waves at successive moments in time while quantifying the continuous signal as a series of discrete values so it can be stored and manipulated digitally.

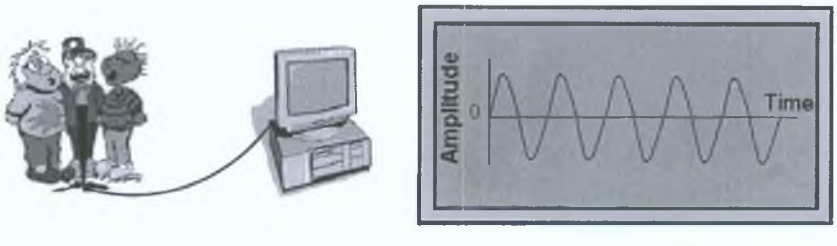


Figure 2.9: Sampling

The parameters of sampling are the *sampling rate* (the number of times per second that the signal is sampled) and the *bit resolution* (number of bits used in each sample) A related parameter is whether the signal is a mono or stereo signal

2 4 1 Sampling Rate

The **sampling rate** is the number of samples recorded per second The more samples that are recorded, the more accurate a reproduction of the original audio that can be generated If enough samples are not recorded per second, the reproduced signal will be less accurate This is known as under-sampling and Figure 2 10 shows the effect of under-sampling a signal

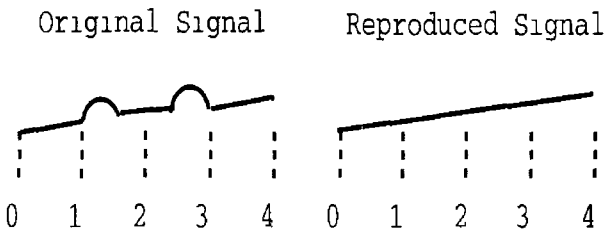


Figure 2 10 Under-sampling

To reproduce sounds accurately, the sampling rate must be at least twice that of the highest frequency that is to be recorded This rule is known as the "Nyqvist theorem" [Kientzle, 98, p24] and exists because for every sample, we have to record both the positive amplitude value as well as the negative amplitude of the wave for each period This is shown in Figure 2 11

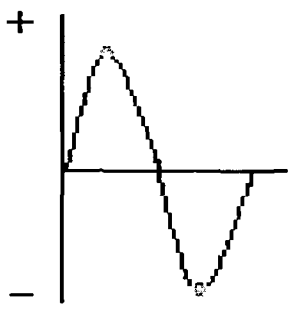


Figure 2 11 Recording the amplitude of the crest as well as the trough

For speech quality sound recordings, the sampling rate is 11,025 samples per second which allows for sounds with frequencies from 0 Hz up to 5,512 Hz to be

captured. This covers the frequency range of the human voice which lies between 50 Hz and 5,000 Hz. Telephone lines typically operate at between 300 Hz and 3,000 Hz. CD quality recordings are sampled at a rate of 44,100 samples per second, capturing frequencies in the range from 0 Hz to 22,050 Hz. This is adequate as the average human ear has a frequency response between 0 Hz and 20,000 Hz. All of this illustrates that there is a trade off between sampling rate and the quality of the audio being recorded. Another factor which will influence the quality of digitised audio is the bit resolution which we now examine.

2.4.2 Bit Resolution

The **number of bits** used in each audio sample strongly influences the fidelity and clarity of recorded audio. When using 8 bits per sample, audio can be represented using numbers in the range of -128 to +127, which means the sampling mechanism can only detect a maximum of 256 different loudness levels. Increasing this to 16 bits per sample means the representation range increases to -32,768 to +32,767, allowing for the recording of up to 65,536 different loudness levels. It should be clear from Figure 2.12 that when sampling, a clearer resolution of the original audio can be gained by increasing the number of bits used to hold the sample, that is, it gives a better representation of the continuous pressure waves which means that at playback time a more accurate rendition of the original audio is generated.

Figure 2.12: Increase in fidelity as a result of using more bits when sampling

The quality of recorded audio is a function of the sampling rate and the bit resolution, but sampling is also subject to a quantisation error, an error which is introduced when converting a continuous audio waveform into a discrete binary value. The occurrence of the error is depicted in Figure 2.13 where it should be seen that the quantisation error is the difference between the analogue signal at sampling time and the nearest discrete value it is assigned. Also shown is the fact that as the

number of bits used increases, the size of quantisation error decreases. This can be seen when comparing Figure 2.13 (a) to 2.13 (b). Observe that the distance from the signal's quantised value to what it actually should be is larger in Figure 2.13 (a) than in Figure 2.13 (b).

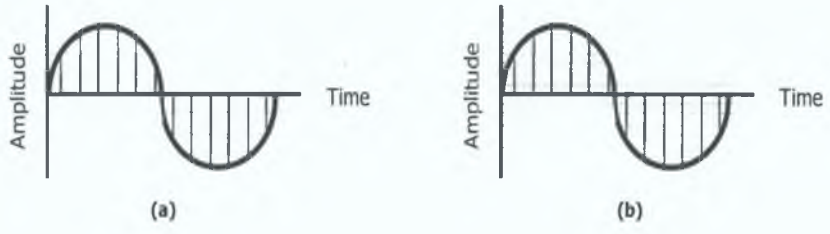


Figure 2.13: Quantisation error

The greater the number of bits available, the smaller the quantisation error. In the worst case scenario the quantisation error is half the value of the smallest distance between the two intervals. The worst case value for signal to quantisation noise can be calculated as:

$$SQNR = 20log\frac{Vsignal}{Vquantisation-noise}$$

$$= 20log\frac{2N-1}{1/2}$$

$$= N * 20log2$$

$$= 6.02NdB$$

(where N is the number of bits used)

Quantisation errors are very noticeable in quiet audio samples using low bit rates, but digitising devices such as sound cards can overcome this by adding a little noise during the sampling process to help amplify the quiet sounds, otherwise the reproduced audio can end up sounding uneven.

When sampling, the difference between the loudest possible signal and the softest possible signal is known as the *dynamic range*. The quietest possible signal is also

the quantisation error As we saw earlier, the human ear has a dynamic range of 120 dB, that is, it can hear sounds as loud as 120 dB Each additional bit used adds 6 dB to the dynamic range, while also decreasing the quantisation error 8 bit recordings support a dynamic range of 48 dB, 16 bit recordings support 96 dB 16 bits is adequate for consumer applications, as most people find the quality of CDs acceptable It is worth noting though that 96 dB is a theoretical limit of the dynamic range of a 16 bit sound card In practice, most 16 bit consumer sound cards only support a dynamic range of approximately 90 dB This is partly due to production issues with low priced components AM quality radio has a dynamic range of 48 dB, hence the impression that it is of low quality, while conventional cassette tapes are only able to support a dynamic range of 65 dB

2 4 3 Channels

Audio can be recorded in either **mono** or **stereo** If it is acceptable that the recording is of a low quality such as speech then mono may be used and one channel is adequate If stereo quality audio is required two channels are used, one for the left and one for the right, both of which are independent of each other

2 4 4 Aliasing

During the sampling process, distortion can be introduced when any frequencies greater than half the sampling rate are present in the analogue signal, i e with a sampling rate of 22,050 Hz, having frequencies present which are greater than 11,025 Hz This distortion is known as **aliasing** and results in any unwanted frequencies wrapping around In the case where a signal is being sampled with a Nyqvist cut-off of 6 Hz, a 7 Hz signal will wrap around to look like a 5 Hz one The effect of aliasing is shown in Figure 2 14 Once aliasing has occurred it can not be removed To prevent its occurrence in the first place, the signal to be sampled must be subjected to a filtering process, thus removing any frequency components above half the sampling rate

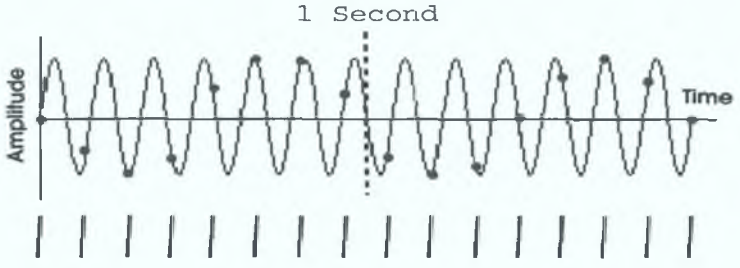


Figure 2.14: A signal subject to aliasing

Low pass filters are used to combat aliasing. A low pass filter allows everything below the cut-off frequency through, while removing everything above the cut-off frequency. Unfortunately the cut-off point is not exact, rather there exists a transition area, from where the filter starts until its desired effect actually kicks in. This is shown in Figure 2.15.

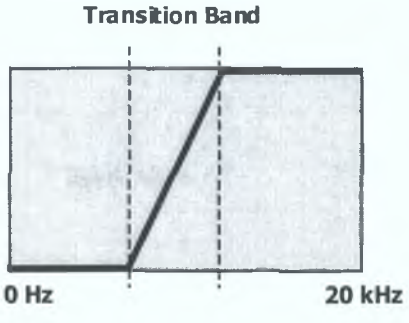


Figure 2.15: Low pass filter showing transition band

This is one of the reasons why CD quality sampling is done at 44,100 Hz and not 40,000 Hz. (44,100 Hz / 2) - 20,000 Hz = 2,050 Hz transition band. Other storage formats like the Digital Audio Tape record at a sampling rate of 48,000 Hz. This allows for a wider transition band (48,000 Hz / 2) - 20,000 Hz = 4,000 Hz, giving a wider transition area which does less damage when filtering out the higher frequencies.

2.4.5 Clipping

Clipping is another unpleasant form of audio distortion which can be introduced when sampling. Clipping occurs when the amplitude of the signal being sampled

is greater than the maximum amplitude that the Analogue to Digital converter supports. There is a cut-off point when the maximum amplitude is reached and any sounds louder than this will be assigned the maximum loudness value, resulting in a flat sounding sample, a bit like what is heard when someone speaks too close to a microphone. Clipping is depicted in Figure 2.16 using an increasing sine wave until the maximum amplitude is reached. Once clipping has occurred it cannot be rectified.

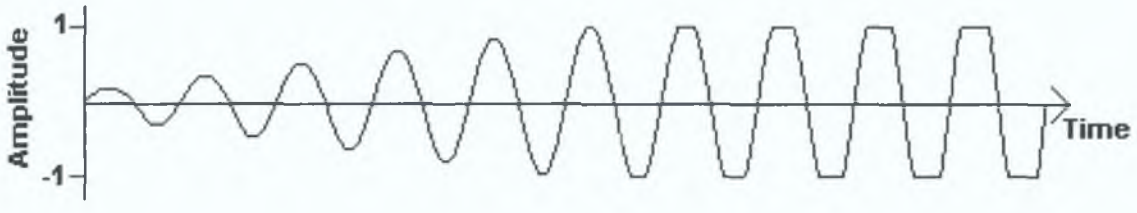


Figure 2.16: A signal subject to clipping

2.5 Storing and Compressing Digital Audio

2.5.1 Pulse Code Modulation

Pulse Code Modulation (PCM) is the format that raw audio or the waveform data is stored as. Storing CD quality PCM audio without any compression occupies 176,400 bytes per second, 10,584,000 bytes per minute or about 635 MB per hour. The huge demand to store and transmit audio in multimedia applications and on the Internet has led to the development of codecs (compressors and decompressors) reducing these storage requirements. Codecs are implemented in hardware and/or software and are used to compress raw audio for storage or transmission and then decompressing it back to raw audio before playback. Compression can be lossy or loss-less, that is, the reproduction loses some of the data or it is exactly the same as the original. Traditional loss-less compression methods developed for general file compression (Huffman [Huffman, 52], Lempel-Ziv [Lempel et al., 77] etc.) are not efficient for compressing audio because audio files do not usually have many repeated sequences such as are present in text files for which general file compression algorithms work well. Even two short snippets of audio which may sound identical

can have substantially different waveform values

The goal of lossy compression is to remove irrelevant data from the original audio samples, thus saving space, but in a way which protects the perceived quality of the audio so when it is played back it sounds like the original There is a trade-off between file size and the quality of playback of lossy compressed audio, as the more an audio file is compressed, the greater the loss of fidelity Different methods are available for implementing lossy compression including techniques such as bit rate reduction and sub-band coding with the removal of irrelevant information based on psychoacoustic principles Some of the different types of loss-less compression methods available include speech compression, non linear Pulse Code Modulation, Differential Pulse Code Modulation, Adaptive Differential Pulse Code Modulation and Predictor based compression

2 5 2 Speech Compression

Speech compression can be achieved using a number of methods One of the simplest is silence detection and can lead to the compression of up to 50% [Kientzle, 98, p45] of the original data, by storing silence as simple codes whenever silence is detected Another method of compressing speech is to create a mathematical model of the human vocal tract and to build an analysis engine that can create parameters to model the given speech This type of encoding of voice audio aims to transmit only the minimal amount of information such that when it is synthesised, it is perceived to be accurate Two common implementations of voice coders or vocoders are Linear Predictor Coding (LPC) and Code Excited Linear Predictor (CELP)

LPC achieves a compression rate as low as 2 4 kbps [Tucker et al , 99] with the use of a model of the human vocal tract and parameterismg the spoken audio over the model Only the parameters of the model are then transmitted The voice is regenerated at the receiving end using the vocal tract model and the transmitted parameters, synthesising the spoken audio CELP is similar to LPC, as it performs the same linear prediction, but it also transmits an error value CELP can achieve bit rates as low as 4 8 kbps [Atal et al , 84]

2 5 3 Non-linear PCM

Linear coding of a continuous signal to its binary representation encodes audio samples on an equidistant scale, that is, the amplitude values are quantised in equal sized steps Non-linear encoding schemes have the advantage that they can offer the lower amplitudes greater resolution, although at the cost of the louder ones This is also the way humans perceive audio We are more sensitive to small changes at low volumes than similar changes at high volumes Imagine a person talking quietly, nearly whispering On a linear scale, it is conceivable that a quietly spoken passage will experience difficulties during the quantisation process Encoding the same passage non-linearly offers the ability to increase the resolution for the quieter portions while at the same time maintaining a large enough range to encode the higher amplitudes Figure 2 17 shows the basic difference between linear encoding and non-linear encoding It should be clear from this figure that non-linear encoding uses a non-uniform amplitude scale while linear encoding is undertaken on a equidistant one The most common implementations of non-linear encoding are μ-Law and A-Law, which are both international telephony encoding standards with North America and Japan using μ-Law encoding while Europe and the rest of the world use A-Law encoding [Kientzle, 98, p82]

A-Law encoding divides the the input range into 7 segments The smallest segment contains 32 intervals, while the remaining 6 each contain 16 intervals The interval size doubles from one segment to the next The first segment has an interval size of 2, with 32 intervals The last segment has an interval size of 128, with 16 intervals Figure 2 17 (a) illustrates the concept of non-linear encoding showing three segments with four intervals in each

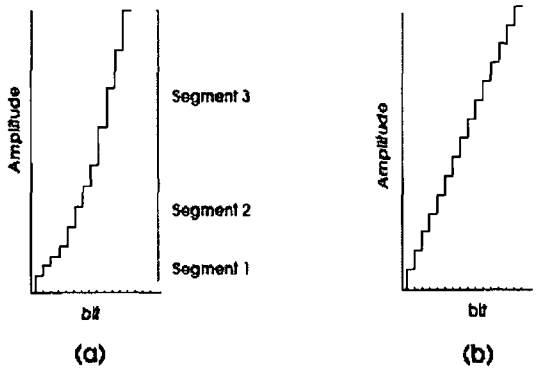


Figure 2 17 Non-linear versus linear encoding

Using 12 bits to encode audio generates a range of 2^{12} or 4096 loudness levels 8 bit A-Law encoding can also achieve this range through the use of the non-uniform interval size Observing the addition of the interval ranges from Table 2 2, A-Law achieves a total range of 4,096, equivalent to the 12 bit linear PCM

Segment	# Intervals	Interval Size	Interval Range
1	32	2	64
2	16	4	64
3	16	8	128
4	16	16	256
5	16	32	512
6	16	64	1024
7	16	128	2048

Table 2 2 A-Law segments

2 5 4 Differential PCM

Differential PCM (DPCM) is a method that only stores the difference between contiguous PCM samples in an audio file, thus reducing the number of bits needed for every recorded sample The difference between contiguous samples at a high sampling rate can be small as a sound may not change that much in 1/44,100 of a second, hence DPCM is most effective at high sampling rates The same data can be quantised, using a fewer number of bits Typical savings of DPCM compresses 64 kbps down to 56 kbps [Kientzle, 98, p98]

2 5 5 Adaptive DPCM

Adaptive Differential Pulse Code Modulation (ADPCM) compresses PCM signals with varying power over time It is more efficient than DPCM as the quantiser is not fixed This means that it adapts the number of bits needed based on the changing amplitude of the input signal The power when sampling speech can vary a lot during the course of a sentence ADPCM adapts the quantiser to respond appropriately There are different implementations of ADPCM which are governed by the International Telecommunication Union (ITU) and offer a variety of compression rates Basic ADPCM achieves a 32 kbps compression rate [ITU, 90] but it also allows for the encoding of higher quality speech (between 50 Hz and 7 kHz) by splitting the input into two sub-bands and implementing ADPCM on the two sub-bands, incurring an overall bit rate of 64 kbps

2 5 6 Predictor Compression

Predictor-based compression works by predicting what the next bit of data will be based on what has gone before and both the compressor and the decompressor predict what the next sample should be The compressor predicts the next sample and if it is correct it outputs a bit indicating that it is correct If it is wrong it outputs a "wrong" bit and the actual sample With a well designed predictor the output will be much smaller than the input

2 5 7 Perceptual Compression

Perceptual-based compression is a lossy compression method which works by identifying and removing information which is deemed perceptually irrelevant based on psychoacoustic principles which include frequency and temporal masking Temporal masking is the phenomenon where, if we hear a loud sound and it subsequently stops, a short time transpires before we can hear a soft tone nearby Frequency masking occurs when a lot of signal energy is present at one frequency with lower energy at nearby frequencies the ear cannot hear the lower frequencies

2.5.8 Sub-Band Coding

Sub-band coding is achieved by compressing different bands of frequencies. It does this by exploiting well-known facts about human hearing. Human hearing is sensitive from 20 Hz to 20 kHz, but it is most sensitive between 1.5 kHz and 5 kHz. The human ear is more sensitive to some frequencies and less sensitive to others and exploiting this fact allows for the preservation of sub-bands where human hearing is most sensitive and the compression or removal of sub-bands where human hearing is not as sensitive. Sub-band coding can also make use of frequency compression within bands.

2.5.9 Bit Reduction

Bit reduction can also be used when trying to reduce the size of sampled audio. It is not always necessary to use the full number of bits available when storing an audio sample. The problem with reducing the number of bits is that it raises the noise floor. The noise floor is that background noise or "hum" which can be heard in low quality recordings, which exists as noise within the system. This noise is normally masked by sound but as the sound level drops the noise floor becomes audible. Reducing the number of bits also increases the quantisation error. As we saw earlier (Section 2.4.2), the quantisation error is introduced during sampling and is a result of the quantisation process, the rounding of a continuous value to a discrete one. As we are using fewer bits, the quantisation error becomes more audible. Employing fewer bits also decreases the dynamic range of sound levels that can be stored, again for every bit lost, the dynamic range decreases by around 6 dB. All of these factors have to be taken into account when deciding how many bits to reduce by. The goal of bit reduction is to keep the fidelity of the audio intact while raising the noise floor to just below the audible signal.

2.6 Music File Formats

There are many different formats available for storing music or speech but we concentrate on those developed specifically for music. Further details on these can be

found in [Kientzle, 98] These include AIFF, AU, WAV, VOC, Real Audio, MP3, Windows Media Format, MIDI and ABC for musical stave notation

2 6 1 AIFF

Apple adapted the Electronic Arts Interchange File Format, which is a multimedia file format for their Audio Interchange File Format (AIFF) to store high quality sampled sound for their Macintosh computers AIFF contains uncompressed raw audio data in the PCM format broken up into chunks with an associated common chunk describing properties that apply to all chunks Some of the important fields from the AIFF common chunk are

- sumChannels - Stereo / mono
- sumSampleFrames - Number of samples in total
- sampleSize - Number of bits used (8 or 16)
- sampleRate - Sampling rate, up to 48,000 Hz

The AIFF format supports a number of different chunks in addition to the common chunk The file starts with a Form chunk, containing information about the format of the file and every other chunk, followed by a Format Version chunk Other chunks exist, including a MIDI chunk for embedding MIDI System Exclusive messages, an Instrument chunk that can be used to synthesise sounds, a Marker chunk to point to positions in the file, a Comments chunk and a Text chunk for comments and information relating to author and copyright as well as a Sound Data chunk which holds the raw audio Data stored in an AIFF file is stored in big-endian format Apple later extended the AIFF format with a version called AIFF-C, which is AIFF supporting a number of different compression methods

2 6 2 WAV

Microsoft introduced its own version of the Electronic Arts IFF standard to Windows 3 1 called the Resource Interchange File Format The WAV file format is part of the

RIFF standard WAV is very similar to the AIFF file format comprising different chunks or blocks It is normally used to store PCM samples in blocks with a header for each block WAV also has a common block called the Format block containing information pertinent to the file's playback WAV supports a variety of sampling rates up to 44,100 Hz, with 8 and 16 bit sample sizes and utilising either one or two channels

2 6 3 AU

The AU file format was developed by Sun Microsystems It can store both linear and non-linear sampled audio Employing non-linear storage, it encapsulates a level of compression as it is said that it reduces 12 bits down to 8 by storing audio logarithmically Standard implementations of the AU format uses μ-Law encoding which is also the international standard telephony encoding format μ-Law was covered in Section 2 5 3

The AU format is also a versatile format allowing for 8, 16, 24 and 32 bit linear PCM encoding, as well as 8 bit μ-Law and A-Law It also supports a number of different implementations of ADPCM Despite its versatility, the format is not widely used outside of the UNIX derived operating systems

2 6 4 VOC

The VOC file format was developed by Creative Labs for use with their early sound cards The format is optimised for the Intel platform A VOC file is segmented into a header block followed by data blocks Some of the blocks that VOC supports include uncompressed sound data, silence blocks, repeater blocks and comment blocks

2 6 5 MIDI

During the 1970s and early 1980s, music synthesisers became recognised as legitimate musical instruments Combined with the advances in technology and cost-efficiency, synthesiser technology became affordable to more and more people The push behind MIDI was to be able to layer sounds without the need for expensive

studio equipment This would also allow musicians create music from one source, triggering instruments to play sounds at their request (through the use of messages)

This means of sending messages from one synthesiser to another was standardised as the Musical Instrument Digital Interface or MIDI MIDI is in widespread use today and can also found in a variety of products, including electronic keyboards, synthesisers, drum machines and sequencers

MIDI files are unique among audio files in that they do not actually store sampled audio, rather they store an operational representation of the music MIDI is actually a control system, containing instructions to perform particular commands, such as note on and off, preset changes, events and timing information

MIDI is divided up into three parts the hardware interface, the communications protocol and the distribution format We are only concerned with the distribution format of MIDI, specifically the MIDI file A MIDI file is made up of tracks which contain meta-data and timing information A track can contain up to 16 channels which store information relating to an instrument which is playing MIDI files can play notes from a bank of up to 128 different instruments Three different types of MIDI files exist Type 0, Type 1 and Type 2

A *Type 0* MIDI file consists of a single track with the tempo and time signature information included in the track A *Type 1* MIDI file contains multiple tracks, with the tempo and time signature information only included in the first track and the synthesiser must combine the data into an event stream before it is synthesised Finally, a *Type 2* MIDI file contains multiple tracks, with the tempo and time signature information included with each track The basic structure of the different types of MIDI files are shown m Figure 2 18 The 16 channels are included in the track data

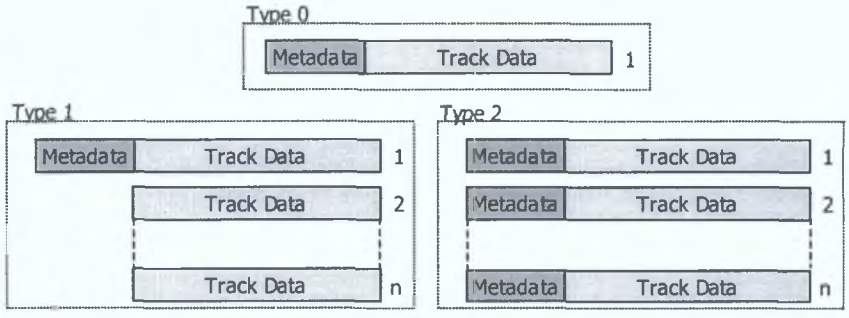


Figure 2.18: The different types of MIDI files

2.6.6 ABC

ABC is an ASCII-based musical notation language. It was designed to notate primarily Western European folk and traditional tunes which can be written on one musical staff in standard classical notation. One of its discerning features is that it is one of the few methods of encoding music which is readable by both humans and computers. Other advantages of encoding music in ABC is that it can be easily ported to other formats e.g. MIDI. It is a format which easily can be mailed or posted to web sites for discussion and distribution because it is easy to read.

2.6.7 MP3

MP3 is the popular name for the MPEG-1 layer 3 standard. Currently it is the most popular way to transmit audio files over the Internet and for the home storage of digitised music collections. With the advent of Napster and its subsequent closure, consumer-generated distribution of copyright music in MP3 is forcing the recording industry to re-think its traditional distribution channels.

The MP3 codec performs lossy compression on PCM audio samples, that is, the reconstructed samples are different to the original, although perceptually, they sound the same. Compression is achieved using sub-band compression and bit reduction and Huffman encoding. The MPEG-1 Layer 3 standard allows compressed audio at bit rates between 32,000 bps and 320,000 bps, compared to 1,400,000 bps for CD quality PCM, thus achieving rates of compression from 2.7 to 44. MP3 compression at about 12:1 or 112,000 bps is indistinguishable from CD quality audio for

most people It is worth noting though that audio compressed to a rate of 32 kbps is of poor quality and a listener would not get much enjoyment from audio compressed and decompressed at this rate The MP3 encoding and decoding process is illustrated in Figure 2 19 It feeds PCM samples to a polyphase filter bank with 32 equal width frequency sub-bands, while simultaneously passing them through a psychoacoustic model to determine Signal-to-Mask ratios for bit reduction The next step uses the Signal-to-Mask ratios to calculate the number of bits required to represent the signal without raising the noise floor so that it becomes audible and thus degrading the quality of the sound The resulting data is then compressed using the Huffman encoding algorithm The compressed MP3 data can then be stored, played or streamed

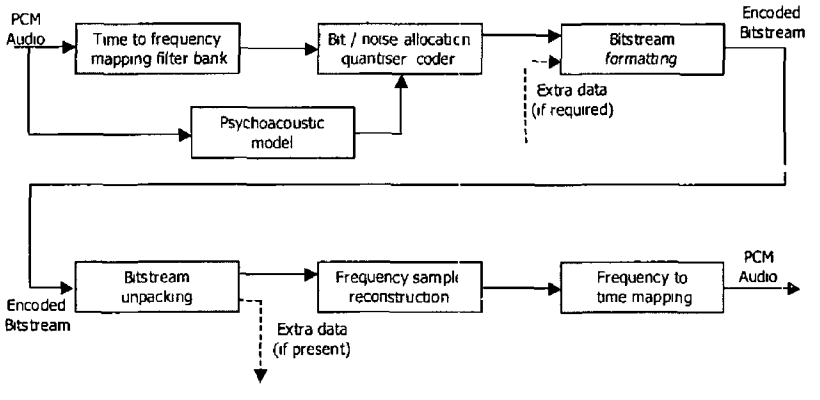


Figure 2 19 MP3 encoding and decoding process
Reproduced from [Pan, 95]

Decompression is the reconstruction of the signal back to the perceptually equivalent original and works by first unpacking the bitstream, reversing the Huffman encoding, reconstructing the frequency bands and then converting the frequency information back into time domain PCM samples

Using MP3 as format for music information retrieval may lead to problems whereby some inaudible frequencies may disappear, although a detailed discussion on how this impacts on music information retrieval is beyond the scope of this dissertation We present some results relating to the use of MP3 as a format for music information retrieval in Chapter 6

2.7 Streaming Audio

We have looked at the different file formats employed for recording audio and have seen how raw audio files with high bit-rates can be compressed without any perceived loss in quality Now we will take a look at some common file formats for streaming audio Streaming audio over the Internet has become very popular in recent years and many radio stations are now providing some form of streaming of their traditional analog based content Some stations have increased the amount of content they provide with supplementary channels and other Internet only radio stations have sprung up P4 [P4, 02], the largest independent radio station in Norway streams its live stream plus three additional content specific channels Radio 1 Oslo [minradio, 02] streams Internet channels only, each of different genres of music The use of the Internet allows traditional radio stations to bypass government restrictions and reach a potentially world-wide audience for a relatively low cost Streaming digital audio is an expanding and lucrative market The importance of open standards is underestimated as major players position themselves to take as much of the market as they can

2 7 1 RealNetworks

The RealAudio format was developed as a method for compressing and delivering voice over low bandwidth transmission lines Early versions used the GSM 06 10 standards and the Code Excited Linear Predictor algorithm RealNetworks then developed their patented plugable codecs for music as well as for video Figure 2 20 shows RealNetworks "One" media player which can be used for streaming audio and video as well as playing multimedia files

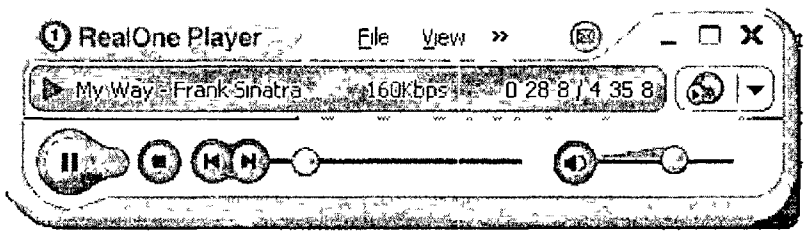


Figure 2 20 RealNetworks "One" player

2 7 2 Windows Media

Whilst a lot of the implementation details of the Windows Media format are unknown, from information based on the patents held by Microsoft Research, Windows Media encoding is based on lapped orthogonal vector quantisation, that is, sampled audio is encoded using a non-scalar vector quantiser producing transform vectors The inverse is applied for the decoding process Microsoft are further extending their Media player and the Windows platform with digital rights management in an attempt to bolster confidence in their format as a secure method of digital music distribution The MS Windows Media encoder allows for the compression of 44,100 Hz, 16 bit stereo encoded PCM at rates from 48 kbps to 160 kbps Figure 2 21 shows the MS Media Player which can be used for streaming audio and video as well as playing multimedia files

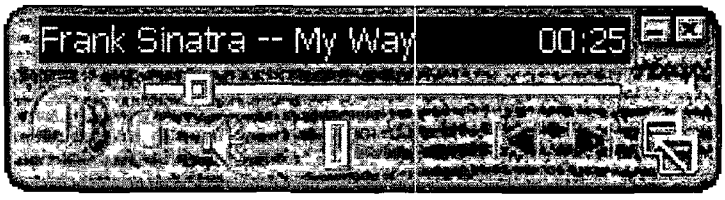


Figure 2 21 MS Media Player

2 7 3 MP3 Streaming

Streaming MP3 has become a popular alternative to both Windows Media and RealNetworks solutions The quality of the audio streams is that of normal MP3 Clients that can be used for MP3 streaming include xmms, x11Amp and winAmp Figure 2 22 shows the X-window based open source audio player, xmms, streaming content from 1FM in Molde, Norway [1Fm, 02] The main advantage of MP3 is that it is a format that computer users are aware of and it has been openly implemented on many platforms

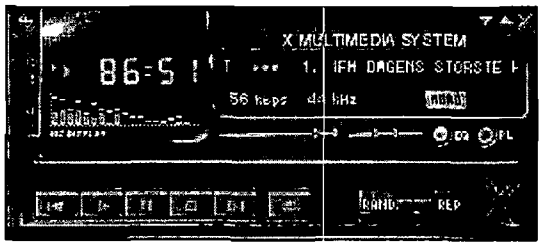


Figure 2 22 xmms audio player streaming MP3

2 7 4 OggVorbis

OggVorbis [Ogg, 02] is a multi platform open source audio streaming codec Its creators goal is to deliver a general purpose codec for mid to high quality audio sources (sampling rates from 8 kHz to 48 kHz with 16 bits and more) which are patent and royalty free The codec is released under a BSD [BSD, 02] style licence Plugins are available for most audio players including xmms, winAmp, x11Amp, Sonique and FreeAMP BBC [BBCOGG, 02] is testing OggVorbis streams of two of their popular channels, Radio 1 and Radio 4 OggVorbis has the opportunity to become a major consideration in the ever increasing audio streaming market as its codec has been implemented for most operating systems, UNIX, Linux, Windows, Macintosh, and BeOS The creators of OggVorbis are currently reviewing an implementation of an open source video codec

The thing that all of these file formats have in common is that they compress audio in order to reduce bandwidth requirements for transmission over the Internet but there is a trade-off between compression and perceived sound quality The different formats have their own methods of implementing their codec's but they have similar performance

2.8 Summary

This chapter presented an introduction to some of the basic properties of sound, how they relate to our hearing and the associated implications for the storage and manipulation of digital audio We also looked at how audio has to be sampled to be stored and some of the issues governing the sampling process We then looked at

storage and compression and observed how compression algorithms have evolved to reduce the high bit rates needed for raw audio i e MP3 reduces a 1 4 Mbps audio file down to 128 kbps with virtually no loss in its perceived sound quality We also observed that music sharing has become a consumer driven reality, posing challenges for the record industry We looked at the file formats available for storing music, in raw, compressed and in notation form Finally, we covered the importance of digital music streaming with a look at the most popular Internet music streaming clients In the next chapter we will take a look at the content of digital audio, in particular the different methods of gaining access to its underlying structure, after which, we will review some of the music information retrieval systems in operation, both past and present

Chapter 3

Technical Background to Digital Music Analysis

3.1 Introduction

This chapter is divided up into 2 sections In the first section we build on the audio fundamentals from the previous chapter and we will review methods for generating frequency spectra and issues associated with their generation We will take a detailed look at Fourier transforms and how they can be implemented to run quickly We will also review other popular methods which can be used to gain access to frequency information including filters and wavelets This provides the context for our explanation of music and musical notes and the differences between notes in terms of audio waveform frequencies In the second part of the chapter we will observe how we can represent the notes played in music as strings using interval or contour, followed by a review of the approaches to string matching, both exact and approximate

3.2 Frequency Analysis of Digital Audio

Before any form of retrieval can be performed on audio information, a deep analysis has to be performed in order to gain access to the actual content of the audio Audio exists simultaneously in two domains, the *time* domain and the *frequency*

domain The time domain presents us with limited information upon which we can build a content based information retrieval system, however it is intuitive to us that the frequency domain is where content-based retrieval should be performed The reasons for this will become clear as we progress through this chapter

3 2 1 Time Domain

The time domain does not present us with much useful information that can be used for retrieval, rather it shows us how the loudness of the recorded audio varies over time Figure 3 1 (a) shows a time domain representation of a 12 second piano accompaniment What is being plotted is how the amplitude varies over time Observing the figure, we can see when a note starts to play and how the power of the note dissipates over time until another note begins to play We cannot infer the pitch of the accompaniment from this time versus amplitude based representation

Figure 3 1 (b) shows part of the same song where the waveform is more defined The time domain uses a series of quantised discrete values to represent the continuous waveform of a sound (Quantisation of a continuous signal was covered in Section 2 4 2) The time domain is useful for editing sounds, copying and pasting sections, overlapping, and also for scaling sounds, for example, making them louder / quieter and fading parts of a sound in and out

3 2 2 Frequency Domain

A frequency domain representation of a number of audio samples is presented through the use of frequency spectra, which quantify the amplitude of any frequencies present in a sampled sound A frequency spectrum is shown in Figure 3 2 Here we can observe that we are plotting frequency versus amplitude for a sound at a given time, while in the time domain we plot amplitude versus time We use a number of frequency spectra generated contiguously to observe how a song's frequency components vary over time The figure illustrates that the sound under analysis has a single frequency component

Meaningful musical analysis has to use the frequency domain in some form

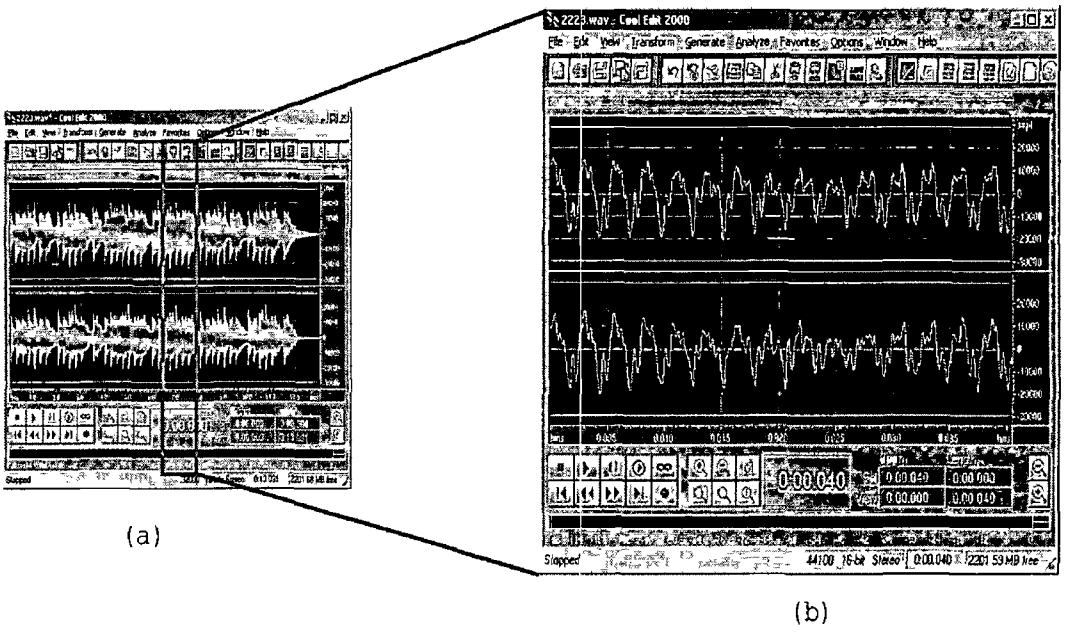


Figure 3 1 Time domain representation of audio

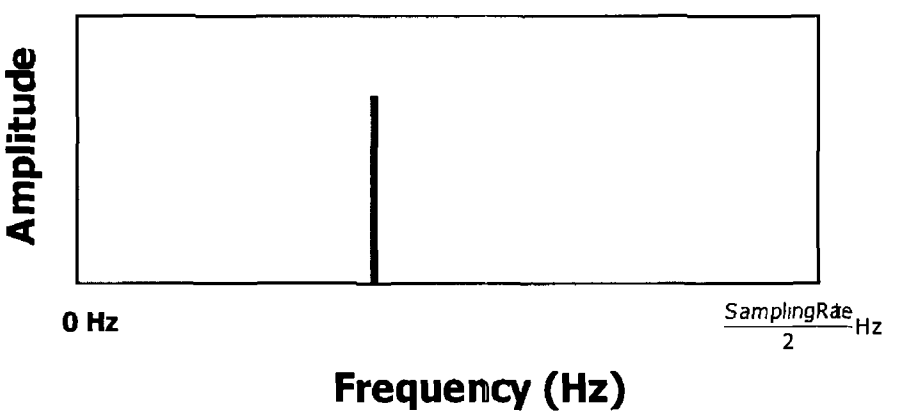


Figure 3 2 A frequency spectrum

Figure 3 3 shows a frequency spectrum of the same piano segment shown in Figure 3 1 The pitch contour of a song is the change in pitch throughout the song (Pitch is covered in Section 2 2 2) Under the frequency domain, the pitch contour of the melody in Figure 3 4 becomes clear It can be seen that the first note played is lower than the second, the third has the same pitch as the second and so on This pitch contour is not clear from Figure 3 1, which shows really only the onset of a note and how the power dissipates, followed by another note onset etc

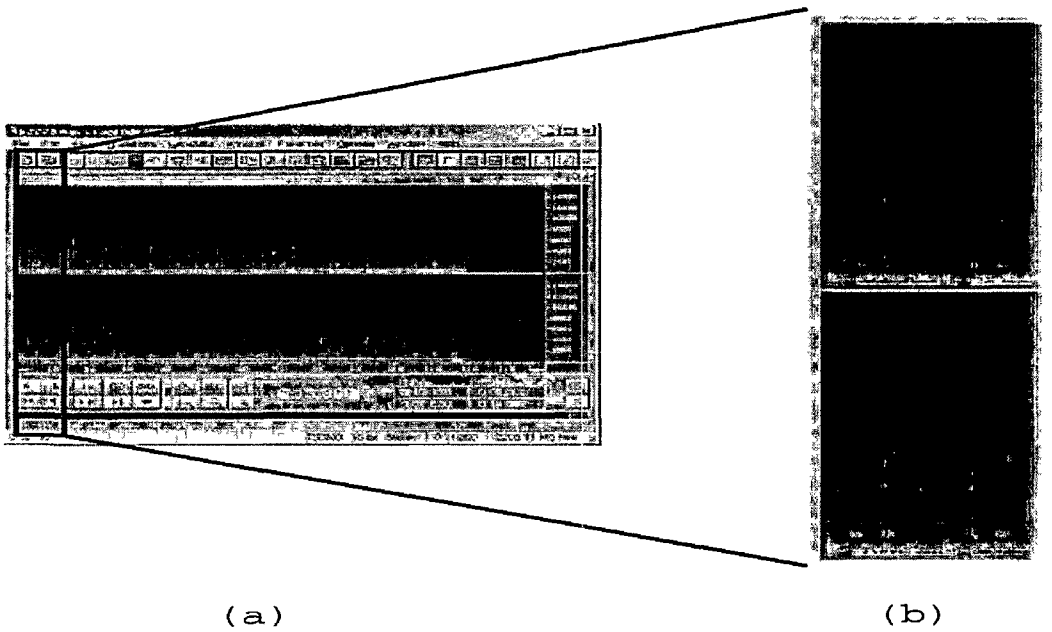


Figure 3 3 Frequency domain representation of audio

A frequency transform takes a time domain signal and generates a frequency spectrum for that signal A frequency spectrum from 0 Hz up to half the sampling rate in Hz can be generated for a sampled sound using a frequency transform For example, observe Figure 3 4 On the left-hand side we have a signal representing a sampled sound in the time domain This is demonstrated by a sine wave with a frequency of 4 Hz, that is, the signal repeats 4 times every second Applying a frequency transform on this sound generates the output on the right, a frequency spectrum with all the frequencies (covered by the transform) and their corresponding amplitudes This is a simplified example, but serves to demonstrate the principle that a transform is used to generate a frequency spectrum

A frequency spectrum is divided up into a number of bins with each bin repre-

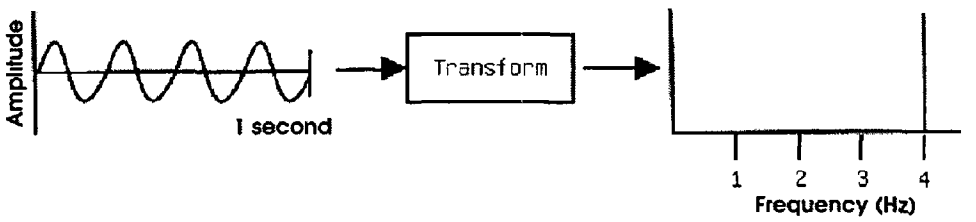


Figure 3 4 A frequency transform

senting a range of frequencies The greater the number of bins, the more fine grained the spectrum This is shown in Figure 3 5 where we can see a relationship between the information obtained from a spectrum and the number of bins used to hold the spectrum The more bins that we choose to use, the more informative the spectrum Observing, the spectrum on the left we can see there is a strong amplitude presence between 2,500 Hz and 5,000 Hz Comparing this to the spectrum on the right, we can see that the bulk of this amplitude actually lies between 2,500 Hz and 3,750 Hz, with the remainder between 3,750 Hz and 5,000 Hz We can also observe the bulk of amplitude between 7,500 Hz and 10,000 Hz actually resides between 7,500 Hz and 8,750 Hz If we were to continue with this process we would end up with a more detailed spectrum

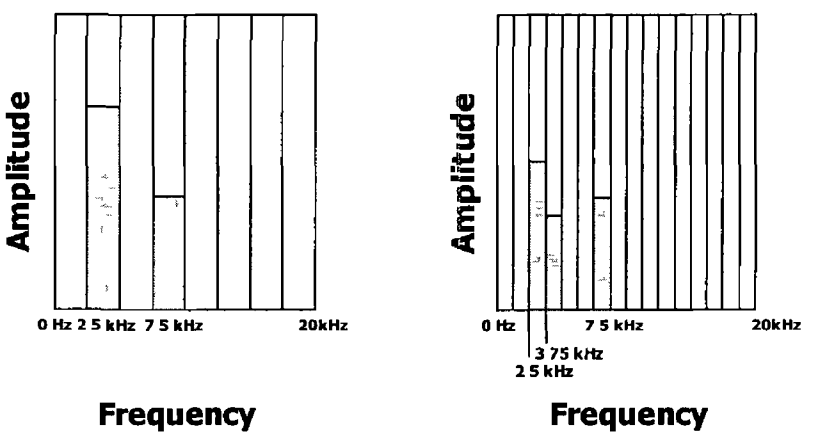


Figure 3 5 Frequency spectrum bins

At the heart of frequency transforms is the use of sine and cosine waves A complex wave can be made up of a number of simpler sine waves Figure 3 6 shows how by combining 2 simple sine waves a third complex wave can be generated This can also be viewed in reverse, where a complex wave is really only the combination of a number of simple sine waves The measurement of the amplitudes of any sine waves

which are present in a sampled sound is the principle upon which frequency analysis is built Frequency transforms are basically a way of expressing a complicated wave as a function of a number of simpler waves

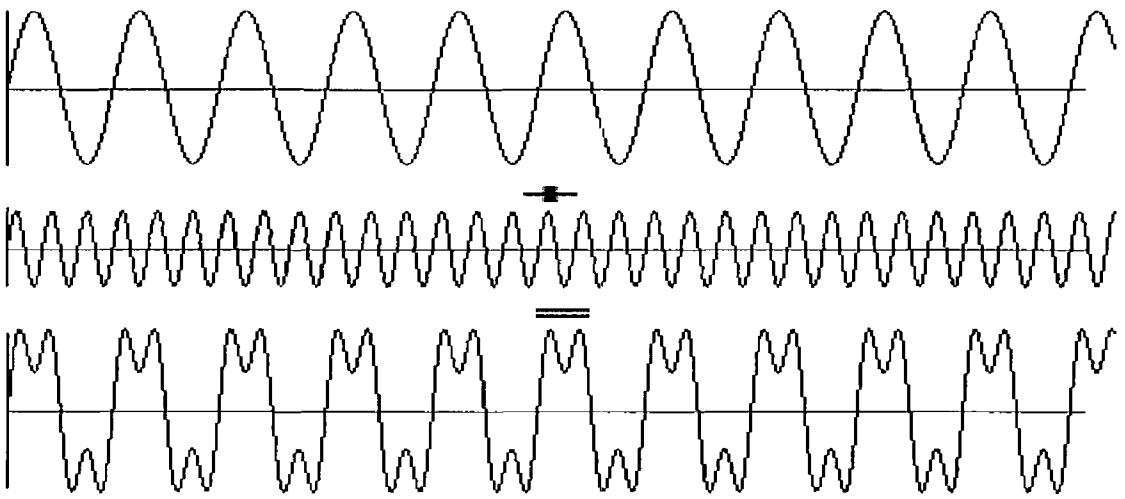


Figure 3 6 A wave made up of a number of simple sine waves

Other than content-based indexing, an example of where one would use the frequency domain would be to remove clicks or hisses which may be present in an analog recording

Before frequency spectra are generated through the use of a transform, they must first be "windowed" to ensure that the introduction of noise as a side effect of the transform process is limited

3 2 3 Windowing

Before analysis of a sampled signal can be undertaken, the start and end points of that signal must not have any unwanted sharp transitions, as such transitions can be regarded as noise Figure 3 7 (a) shows a sampled sound with a sharp jump at the beginning and at the end The sampled signal can not be analysed reliably unless the signal converges to zero at both end points Therefore, before processing it must be windowed, that is, the transitions at the beginning and at the end of the window are smoothed, minimising the introduction of noise A windowed signal is shown in Figure 3 7 (b) This convergence is achieved by using a weighting function One such function is the Hamming Window [Foote, 99b] which uses the formula

$$0.56 - 0.46cos(\frac{2\pi t}{samplesize-1})$$

A Hamming function smooths out the signal at both ends, with little effect on the frequency. Other windowing functions exist including Hanning, Blackman, Triangle, Rectangle, Kaiser and Blackman-Harris [Harris, 78]

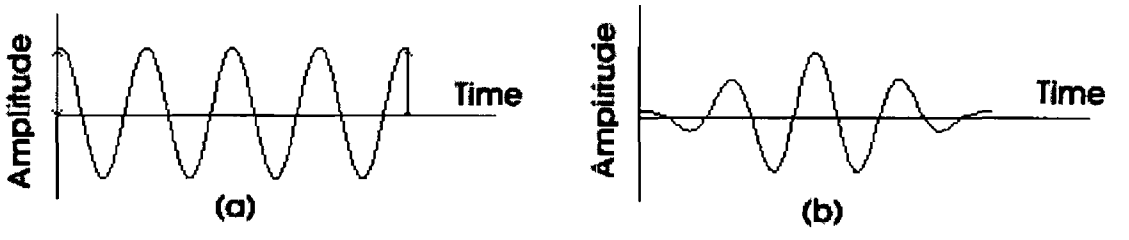


Figure 3.7 Effect of windowing a signal

3.2.4 Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is a method of decomposing a signal into the weighted sum of a number of waves. The principle idea behind the DFT is wave **multiplication** and **addition**. If two sine waves of the same frequency and phase are multiplied and the resulting product samples are added together, then the resulting addition will be a positive number as any negative amplitude values in both waves when multiplied will result in an overall positive value. This is shown in Figure 3.8 (a). Similarly, this can also be applied to a more complex wave. If a complex wave contains the test frequency, when each value in the complex wave is multiplied by each value in the test frequency and added, the overall result will be positive. If the waves are of differing frequencies, when multiplied, the sum of the resulting product samples will be zero. This is because the positive amplitude values will cancel out with the negative amplitude values. This is shown in Figure 3.8 (b).

This method of multiplication and addition can be used to measure the amplitude (or loudness) of *one* frequency component at a time. To generate an entire frequency spectrum with a DFT, we would have to call the DFT for each frequency component we wish to analyse.

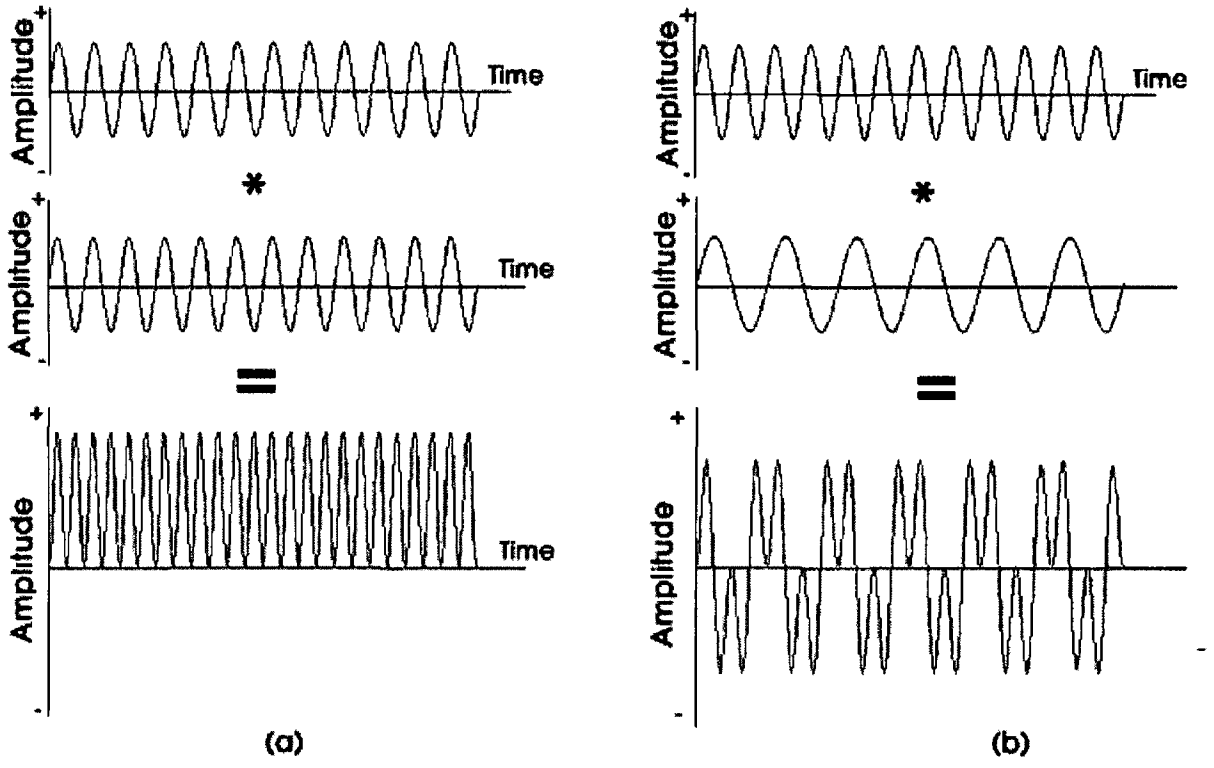


Figure 3 8 Multiplying two waves

As we saw earlier, we can measure the amplitude of a particular frequency in a signal by multiplying two waves and adding the product samples The formula for the DFT to calculate the amplitude A^r of a given frequency k [Kientzle, 98] and [TCLEX, 95] is

$$A_k^r = \sum_{n=0}^{N-1} s_n cos\left(\frac{2\pi nk}{N}\right)$$

where

N represents the number of time domain samples

n is the sample number $(0 \geq n < (N-1))$

s_n represents the amplitude of sample n

k is frequency under investigation

A^r indicates that this is a real number (rather than a complex one)

However, this only gives us the amplitude Fourier transforms can indirectly calculate the phase (phase is covered in Section 2 2 1) of a wave as well This is achieved by using complex numbers and the cosine and sine functions

$$A_k = \sum_{n=0}^{N-1} s_n cos\left(\frac{2\pi nk}{N}\right) - \imath\ s_n sin\left(\frac{2\pi nk}{N}\right)$$

where

$\imath = \sqrt{-1}$

$cos(x) + \imath\ sin(x)$ is a complex number which can be used to represent a point on a complex plane with the x coordinate calculated using $x = Re\{A_k\} = s_n cos\left(\frac{2\pi nk}{N}\right)$ and the y coordinate calculated using $y = Im\{A_k\} = -s_n sin\left(\frac{2\pi nk}{N}\right)$ The magnitude of a complex number, hence the amplitude of the frequency in question, can be calculated using the formula

$$\text{magnitude} = \sqrt{Re\{A_f\}^2 + Im\{A_f\}^2}$$

Figure 3 9 shows a point on a complex plane It should be clear that the amplitude can be calculated as the length of the hypotenuse The phase of the wave is then the angle between the horizontal line to the hypotenuse The phase can be calculated as

$$tan^{-1}\left(\frac{Im\{A_k\}}{Re\{A_k\}}\right)$$

The formula

$$A_k = \sum_{n=0}^{N-1} s_n cos\left(\frac{2\pi nk}{N}\right) \text{ - } \imath\ s_n sin\left(\frac{2\pi nk}{N}\right)$$

can be simplified into a more cryptic but equivalent formula

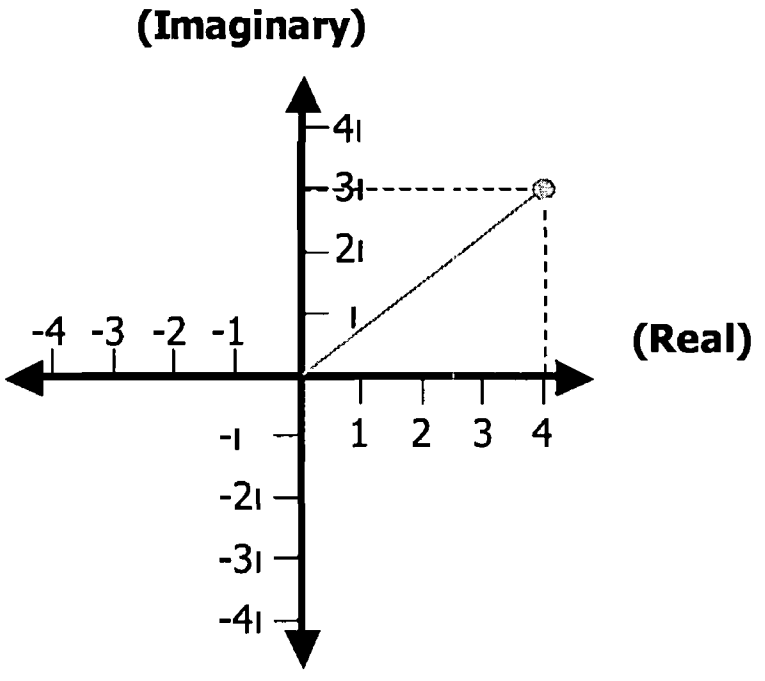


Figure 3 9 Calculating magnitude and phase on a complex plane

$$A_k = \sum_{n=0}^{N-1} s_n e^{\left(\frac{-i2\pi nk}{N}\right)}$$

$$\text{as } e^{-ix} = cos(x) - i\ sin(x)$$

The inverse of a DFT can be applied to the DFT'd data to regenerate the original signal using the formula

$$s(k) = \frac{1}{N} \sum_{m=0}^{N-1} A_k e^{\left(\frac{i2\pi mk}{N}\right)}$$

When undertaking the IDFT, we must divide the summation by $\frac{1}{N}$ This is a side effect of the summation in the DFT resulting in a value which is N times larger than the original

The output of a Fourier transform is mirrored, that is, the real values of the

complex numbers on the right hand side of the spectrum are mirrored values of the left hand side, while the imaginary values are conjugated This phenomenon is shown in Figure 3 10 The real values below N/2 are mirrored above N/2 Not shown are the conjugated imaginary values

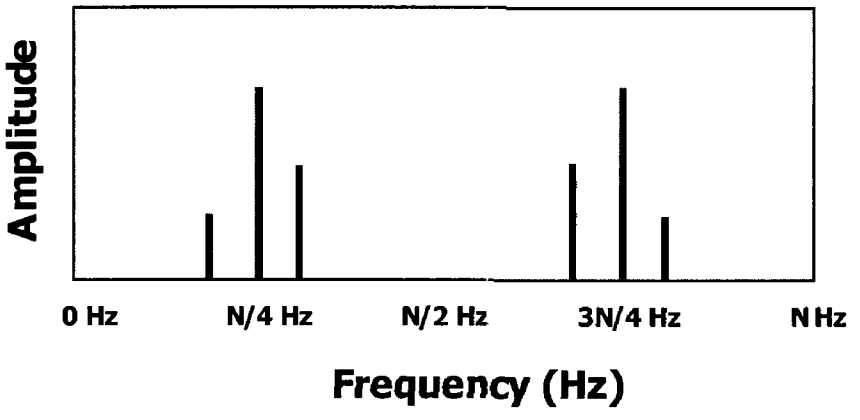


Figure 3 10 Mirrored output of a Fourier transform

The disadvantage of using the Discrete Fourier Transform for spectra generation is that the summation has to be done for every frequency, thus an entire frequency spectrum will require N*N calculations with a complexity of $O(n^2)$

A Fourier Transform assumes that the data is subject to a period of N, that is, the signal repeats after N samples A common implementation of the DFT is the Fast Fourier Transform which can be implemented to run very quickly if the number of samples (N) is a power of 2

3 2 5 Fast Fourier Transform

The Fast Fourier Transform (FFT) [Cooley et al , 65] computes an entire spectrum with a complexity of $O(NlogN)$, by eliminating many of the repeated calculations of the DFT

Remember we are expressing a complex wave decomposed as a number of simpler waves These simpler waves are expressed in terms of sine and cosine waves Observing Figures 3 11 and 3 12 we can draw both a sine wave and a cosine wave from a circle

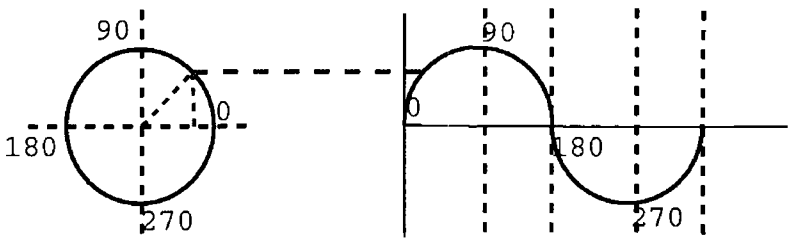


Figure 3 11 Drawing a sine wave from a circle

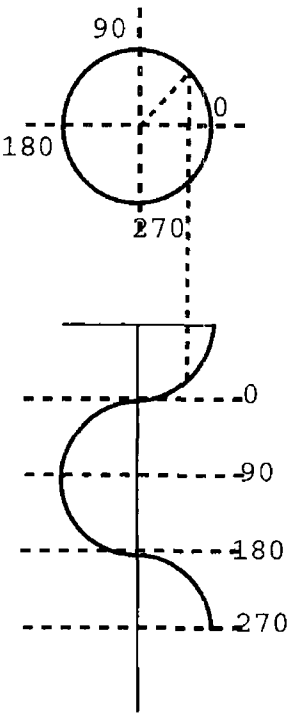


Figure 3 12 Drawing a cosine wave from a circle

If we divide a circle into N segments, each segment is $\frac{360^o}{N}$ For every point on the circle, n$\left(\frac{360}{N}\right)$, then $(n * k)\left(\frac{360}{N}\right)$ is the same point for values of k where n*k modulo N is the same number $4 * 1\frac{360}{8} = 180^o$,$(4 * 3)\left(\frac{360}{8}\right) = 540^o$, and so on, are the same point on a circle This is shown in Figure 3 13

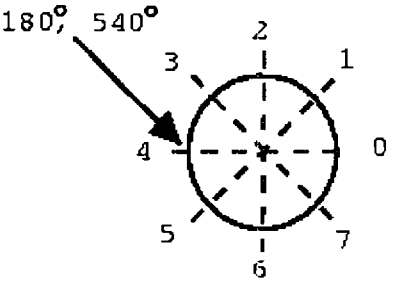


Figure 3 13 Segmenting a circle

Using 2π ($\pi = 180^o$= 3 141592654), we can represent 360^o in radians rather than degrees $\cos\left(\frac{2\pi nk}{N}\right)$ can be used to represent a wave Observing the values k and n, k is the number of times we wish to rotate through the circle, hence is related to frequency If k=5, and if it takes 1 second to rotate through the circle 5 times, then it has frequency of 5 Hz n is the evenly displaced point on the circle and varies from 0 to N-1 We can observe that

$$cos\left(\frac{2\pi(1)(4)}{8}\right) = cos\left(\frac{2\pi(3)(4)}{8}\right) = cos\left(\frac{2\pi(5)(4)}{8}\right) = -1$$

Figure 3 14 shows this What this means is that anytime we have to calculate $cos\left(\frac{2\pi(1)(4)}{8}\right)$ we know that the same answer applies to $cos\left(\frac{2\pi(3)(4)}{8}\right)$ and any other cos function which has the same value when nk is modulo'd with N Using the formula $e^{-ix} = cos(x) - i\ sin(x)$, the above formula is also expressible as $e^{\frac{-i2\pi nk}{N}}$

The next stage in the process is to manipulate values of n and k in the formula We can take n and k out

$$e^{\frac{-i2\pi nk}{N}} = \left(e^{\frac{-i2\pi}{N}}\right)^{nk}$$

as $(e^{2*3}) = (e^2)^3$

It should then be observed that $e^{\frac{-i2\pi}{N}}$ becomes a constant value once N is defined Continuing our discussion on how the FFT is quicker than a DFT requires the repeated presentation of $e^{\frac{-i2\pi}{N}}$ To keep the discussion on the FFT easy to read

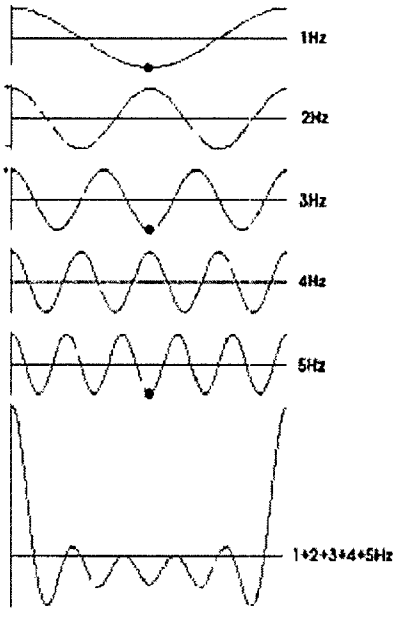


Figure 3 14 Fourier expansion of a wave

and with the fact that $e^{\frac{-i2\pi}{N}}$ is a constant value, we will rename $e^{\frac{-i2\pi}{N}}$ as M This allows us to illustrate the workings of the FFT without getting bogged down in the complexity

Using N=8, we can observe the following computations to calculate an entire frequency spectrum using the DFT

$$
\begin{aligned}
A_0 &= s(0)M^0+s(1)M^0+s(2)M^0+s(3)M^0+s(4)M^0+s(5)M^0+s(6)M^0+s(7)M^0\\
A_1 &= s(0)M^0+s(1)M^1+s(2)M^2+s(3)M^3+s(4)M^4+s(5)M^5+s(6)M^6+s(7)M^7\\
A_2 &= s(0)M^0+s(1)M^2+s(2)M^4+s(3)M^6+s(4)M^8+s(5)M^{10}+s(6)M^{12}+s(7)M^{14}\\
A_3 &= s(0)M^0+s(1)M^3+s(2)M^6+s(3)M^9+s(4)M^{12}+s(5)M^{15}+s(6)M^{18}+s(7)M^{21}\\
A_4 &= s(0)M^0+s(1)M^4+s(2)M^8+s(3)M^{12}+s(4)M^{16}+s(5)M^{20}+s(6)M^{24}+s(7)M^{28}\\
A_5 &= s(0)M^0+s(1)M^5+s(2)M^{10}+s(3)M^{15}+s(4)M^{20}+s(5)M^{25}+s(6)M^{30}+s(7)M^{35}\\
A_6 &= s(0)M^0+s(1)M^6+s(2)M^{12}+s(3)M^{18}+s(4)M^{24}+s(5)M^{30}+s(6)M^{36}+s(7)M^{42}\\
A_7 &= s(0)M^0+s(1)M^7+s(2)M^{14}+s(3)M^{21}+s(4)M^{28}+s(5)M^{35}+s(6)M^{42}+s(7)M^{49}
\end{aligned}
$$

Figure 3 15 DFT calculations with N=8

To make the process easier to follow we will look at the above calculations in a table with A_0 to A_7 on the left and n across the top These values are shown in Table 3 1

We can see from Table 3 1 that there are only 24 different exponent of M calculations We saw earlier that $M^0 = M^N = M^{2N}$, and $M^1 = M^{N+1} = M^{2N+1}$

	s(0)	s(1)	s(2)	s(3)	s(4)	s(5)	s(6)	s(7)
0	M^0	M^0	M^0	M^0	M^0	M^0	M^0	M^0
1	M^0	M^1	M^2	M^3	M^4	M^5	M^6	M^7
2	M^0	M^2	M^4	M^6	M^8	M^{10}	M^{12}	M^{14}
3	M^0	M^3	M^6	M^9	M^{12}	M^{15}	M^{18}	M^{21}
4	M^0	M^4	M^8	M^{12}	M^{16}	M^{20}	M^{24}	M^{28}
5	M^0	M^5	M^{10}	M^{15}	M^{20}	M^{25}	M^{30}	M^{35}
6	M^0	M^6	M^{12}	M^{18}	M^{24}	M^{30}	M^{36}	M^{42}
7	M^0	M^7	M^{14}	M^{21}	M^{28}	M^{35}	M^{42}	M^{49}

Table 3 1 Exponent calculations for the DFT

Therefore, for every nk which we modulo with N, we only need to perform the multiplication once as no matter what value exponent we have, it can always be expressed as an exponent in the range 0 to N-1 Applying modulo N to Table 3 1 produces Table 3 2 This reduces the number of *different* multiplications which have to be carried out The next step is to reduce the number of *actual* multiplications undertaken

	s(0)	s(1)	s(2)	s(3)	s(4)	s(5)	s(6)	s(7)
0	M^0	M^0	M^0	M^0	M^0	M^0	M^0	M^0
1	M^0	M^1	M^2	M^3	M^4	M^5	M^6	M^7
2	M^0	M^2	M^4	M^6	M^0	M^2	M^4	M^6
3	M^0	M^3	M^6	M^1	M^4	M^7	M^2	M^5
4	M^0	M^4	M^0	M^4	M^0	M^4	M^0	M^4
5	M^0	M^5	M^2	M^7	M^4	M^1	M^6	M^3
6	M^0	M^6	M^4	M^2	M^0	M^6	M^4	M^2
7	M^0	M^7	M^6	M^5	M^4	M^3	M^2	M^1

Table 3 2 Exponent values after applying modulo n

The first step of this process is to divide the table into odd and even values of n This is shown in Table 3 3

Mathematically, this process can be expressed using the formula

$$A_k = \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{\frac{-i2\pi}{N}}\right)^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{\frac{-i2\pi}{N}}\right)^{2k(n+1)}$$

which can be reduced to

	s(0)	s(2)	s(4)	s(6)
	0	1	2	3
0	M^0	M^0	M^0	M^0
1	M^0	M^2	M^4	M^6
2	M^0	M^4	M^0	M^4
3	M^0	M^6	M^4	M^2
4	M^0	M^0	M^0	M^0
5	M^0	M^2	M^4	M^6
6	M^0	M^4	M^0	M^4
7	M^0	M^6	M^4	M^2

Even

	s(1)	s(3)	s(5)	s(7)
	0	1	2	3
0	M^0	M^0	M^0	M^0
1	M^1	M^3	M^5	M^7
2	M^2	M^6	M^2	M^6
3	M^3	M^1	M^7	M^5
4	M^4	M^4	M^4	M^4
5	M^5	M^7	M^1	M^3
6	M^6	M^2	M^6	M^2
7	M^7	M^5	M^3	M^1

Odd

Table 3 3 Dividing exponents into odd and even values of n

$$A_k = \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{2\left(\frac{-\imath 2\pi}{N}\right)}\right)^{nk} + \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{2\left(\frac{-\imath 2\pi}{N}\right)}\right)^{k(n+1)}$$

which can be reduced to

$$A_k = \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{\frac{-\imath 2\pi}{\frac{N}{2}}}\right)^{nk} + \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{\frac{-\imath 2\pi}{\frac{N}{2}}}\right)^{k(n+1)}$$

then we factor in the n

$$A_k = \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{\frac{-\imath 2\pi}{\frac{N}{2}}}\right)^{nk} + \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{\frac{-\imath 2\pi}{\frac{N}{2}}}\right)^{nk+k}$$

but recall that $(e^{2+3} = e^2 \ \ e^3)$ so

$$A_k = \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{\frac{-\imath 2\pi}{\frac{N}{2}}}\right)^{nk} + \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{\frac{-\imath 2\pi}{\frac{N}{2}}}\right)^{nk} \ \ \left(e^{\frac{-2\pi}{N}}\right)^{k}$$

and seeing as k does not change with $\sum$, we can move that part outside

$$A_k = \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{\frac{-i2\pi}{\frac{N}{2}}}\right)^{nk} + \left(e^{\frac{-i\,'\pi}{N}}\right)^{k} \sum_{n=0}^{\frac{N}{2}-1} s_n \left(e^{\frac{-i2\pi}{\frac{N}{2}}}\right)^{nk}$$

Now, for both the even values and odd values of n we only have to multiply in M, 16 times (32 operation in total) Instead of multiplying by 8 points of the wave, we now only have to multiply by 4 of them We make up the difference by multiplying the odd samples by a 45^o phase shift (360/8) This is illustrated in Figure 3 16 and can also be seen in Table 3 4

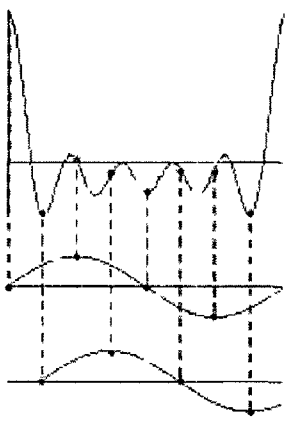


Figure 3 16 45^o phase shift of the odd samples

	s(0)	s(2)	s(4)	s(6)
	0	1	2	3
0	M^0	M^0	M^0	M^0
1	M^0	M^2	M^4	M^6
2	M^0	M^4	M^0	M^4
3	M^0	M^6	M^4	M^2
4	M^0	M^0	M^0	M^0
5	M^0	M^2	M^4	M^6
6	M^0	M^4	M^0	M^4
7	M^0	M^6	M^4	M^2

Even

$\mathbf{M^f}$

	s(0)	s(2)	s(4)	s(6)
	0	1	2	3
0	M^0	M^0	M^0	M^0
1	M^0	M^2	M^4	M^6
2	M^0	M^4	M^0	M^4
3	M^0	M^6	M^4	M^2
4	M^0	M^0	M^0	M^0
5	M^0	M^2	M^4	M^6
6	M^0	M^4	M^0	M^4
7	M^0	M^6	M^4	M^2

Odd

Table 3 4 Multiplying in the first phase shift

Continuing this process, we can save on even more computation Again, we divide Table 3 4 into odd and even values of n, which produces Table 3 5

	s(0)	s(2)
	0	1
0	M^0	M^0
1	M^0	M^2
2	M^0	M^4
3	M^0	M^6
4	M^0	M^0
5	M^0	M^2
6	M^0	M^4
7	M^0	M^6

Even

	s(4)	s(6)
	0	1
0	M^0	M^0
1	M^4	M^6
2	M^0	M^4
3	M^4	M^2
4	M^0	M^0
5	M^4	M^6
6	M^0	M^4
7	M^4	M^2

Odd

M^f

	s(1)	s(3)
	0	1
0	M^0	M^0
1	M^0	M^2
2	M^0	M^4
3	M^0	M^6
4	M^0	M^0
5	M^0	M^2
6	M^0	M^4
7	M^0	M^6

Even

	s(5)	s(7)
	0	1
0	M^0	M^0
1	M^4	M^6
2	M^0	M^4
3	M^4	M^2
4	M^0	M^0
5	M^4	M^6
6	M^0	M^4
7	M^4	M^2

Odd

Even **Odd**

Table 3 5 Further dividing into odd and even

Now, for both the even and odd sets, instead of multiplying by 4 points of the wave, we are now only multiplying 2 of them For the even side, we multiply the odd samples by a 90^o phase shift (360/4) For the odd side, the odd samples are also subjected to the same 90^o phase shift The final values are shown in Table 3 6

	s(0)	s(4)
	0	1
0	M^0	M^0
1	M^0	M^4
2	M^0	M^0
3	M^0	M^4
4	M^0	M^0
5	M^0	M^4
6	M^0	M^0
7	M^0	M^4

M^f

	s(2)	s(6)
	0	1
0	M^0	M^0
1	M^2	M^6
2	M^4	M^4
3	M^6	M^2
4	M^0	M^0
5	M^2	M^6
6	M^4	M^4
7	M^6	M^2

M^{2f}

	s(1)	s(3)
	0	1
0	M^0	M^0
1	M^0	M^4
2	M^0	M^0
3	M^0	M^4
4	M^0	M^0
5	M^0	M^4
6	M^0	M^0
7	M^0	M^4

M^f

	s(5)	s(7)
	0	1
0	M^0	M^0
1	M^2	M^6
2	M^4	M^4
3	M^6	M^2
4	M^0	M^0
5	M^2	M^6
6	M^4	M^4
7	M^6	M^2

Even **Odd**

Table 3 6 Final table for the FFT

This process can be expressed with the formula

$$A_k = \sum_{n=0}^{\frac{N}{4}-1} s_n \left(e^{\frac{-\imath 2\pi}{\frac{N}{4}}}\right)^{nk} + \left(e^{\frac{-\imath 2\pi k}{N}}\right) \sum_{n=0}^{\frac{N}{4}-1} s_n \left(e^{\frac{-\imath 2\pi}{\frac{N}{4}}}\right)^{nk}$$

$$+ \left(e^{\frac{-\imath 2\pi k}{N}}\right) \left(\sum_{n=0}^{\frac{N}{4}-1} s_n \left(e^{\frac{-\imath 2\pi}{\frac{N}{4}}}\right)^{nk} + \left(e^{\frac{-\imath 2\pi k}{\frac{N}{4}}}\right)^{k} \sum_{n=0}^{\frac{N}{4}-1} s_n \left(e^{\frac{-\imath 2\pi}{\frac{N}{4}}}\right)^{nk} \right)$$

Using the FFT we have reduced the number of calculations from 64 calculations to 24 Instead of multiplying by 8 points of the wave, we are now only multiplying by 2 of them, making up the difference using phase shifting We have to undertake the $e^{\frac{-\imath 2\pi nk}{\frac{N}{4}}}$ multiplication only 4 times These $e^{\frac{-\imath 2\pi nk}{\frac{N}{4}}}$ are shown in Table 3 6 as (n=0, k=0), (n=1, k=0), (n=0, k=1), (n=1, k=1) Then using these calculations, they can be applied to all other original pair wise values of n (n=2, n=6), (n=3, n=7), (n=1, n=5) For (n=2, n=6) we have to multiply in a 90° phase shift, for (n=1, n=5) we have to multiply in a 45° degree phase shift and for (n=3, n=7) we have to multiply both a 45° and then a 90° phase shift to make up the difference Diagrammatically, we can view this phase shifting as in Figure 3 17

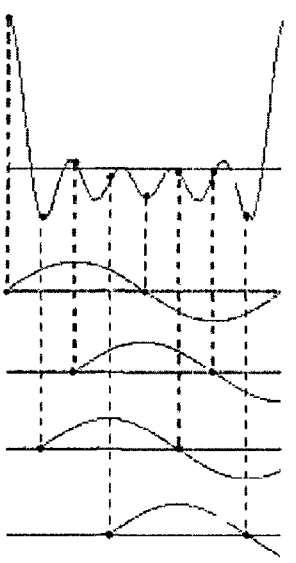


Figure 3 17 Final phase shifting for the FFT

Before we can apply a FFT on an audio file, there are a number of things to consider First, remember that the FFT can only generate a spectrum from 0 Hz

to half the sampling rate Secondly, the length of the transform determines the frequency ranges

For example, using a 1,024 point fast Fourier transform on a number of audio samples, sampled at rate of 44,100 samples per second, the frequency spectrum is divided up into buckets from 0 Hz to 22,050 Hz, with each bucket holding the amplitude in steps of 43 1 Hz A 16,384 point fast Fourier transform on the same samples gives a frequency resolution of 2 723 Hz for each bucket The larger the fast Fourier transform, the finer grained a spectrum can be generated Some example parameters for FFT audio file processing are shown in Table 3 7

Size	Sampling Rate	Step Size	Range
8	11,025	1378 125 Hz	0 - 5,512 Hz
1,024	11,025	10 77 Hz	0 - 5,512 Hz
32,768	11,025	34 Hz	0 - 5,512 Hz
8	22,050	2756 25 Hz	0 - 11,025 Hz
1,024	22,050	21 54 Hz	0 - 11,025 Hz
32,768	22,050	67 Hz	0 - 11,025 Hz
8	44,100	5512 5 Hz	0 - 22,050 Hz
1,024	44,100	43 08 Hz	0 - 22,050 Hz
32,768	44,100	1 34 Hz	0 - 22,050 Hz

Table 3 7 Some parameters for a fast Fourier transform

3 2 6 Short Time Fourier Transform

The Short Time Fourier Transform (STFT) is a method which attempts to evaluate how the frequency content of sound samples change over time Basically it works by breaking the signal into shorter pieces and applying a Fourier transform to each piece It shows how the frequency content evolves through the generation of a number of spectra which are plotted contiguously The STFT is shown in Figure 3 18 As we will see later, this is the method which we will use to generate frequency spectra as part of CEOLAIRE'S melody extraction engine

3 2 7 Filters

Filters can be used to remove certain frequencies from a signal There are three basic type of filters which can be used A Low Pass filter (See Figure 3 19 (a)) cuts

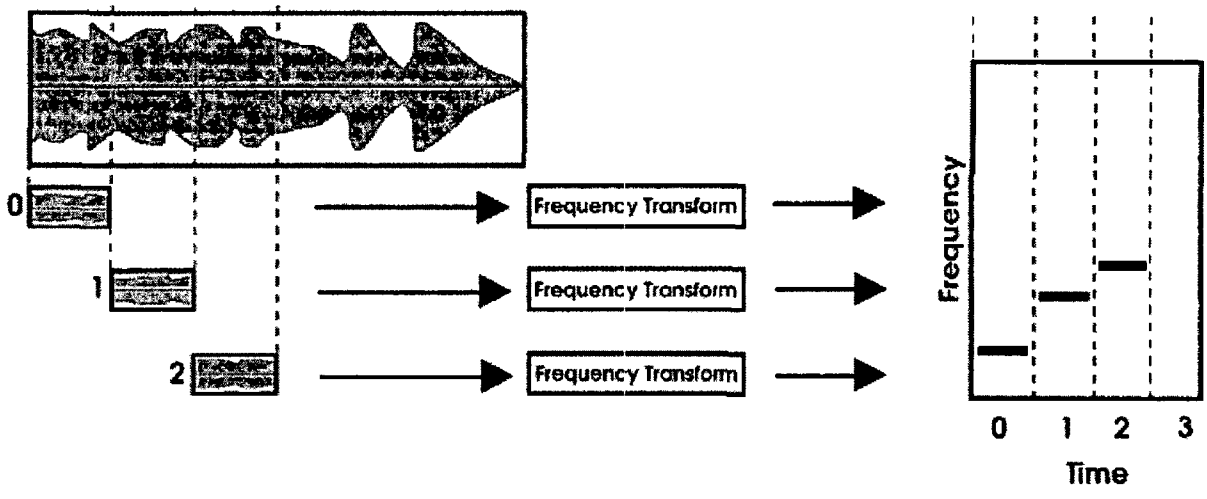


Figure 3 18 Combining single spectra to create a time varying spectrum

out all signals above a certain frequency while allowing all the frequencies below this threshold through A High Pass filter (See Figure 3 19 (c)) removes all frequencies below a certain cut-off point allowing everything above it through Band Pass Filters (See Figure 3 19 (b)) set up a "passable" range, allowing all frequencies within the pass range through, while blocking all others The basic idea behind audio filters is that it multiplies the undesirable frequencies by zero Filters are another method which can be used by CEOLAIRE to filter out musical note information

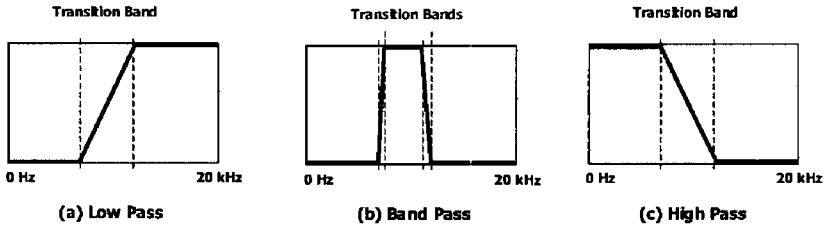


Figure 3 19 Three different types of filters

3 2 8 Wavelets

Wavelets are another method besides the traditional Fourier derived methods for generating frequency spectra Wavelets differ from traditional Fourier analysis in that they introduce the notion of scaling to the spectrum and they approximate functions within a finite domain With Fourier analysis, we assume the signal repeats forever Wavelets work using basis functions Basis functions are the combination of

sine and cosine functions of varying frequencies and amplitudes that are orthogonal, that is, their inner products add up to zero

Unlike Fourier analysis, wavelets have varying sizes for the windows of frequency components Wavelets allow for both short high-frequency basis functions and long low-frequency ones, as well as allowing windows to be constructed with rough edges This allows for better approximations of real world signals Wavelet basis functions are a particular type of basis functions

Wavelet transforms convert a signal into a series of wavelets Signals processed by the wavelet transform can be stored more efficiently than those processed by Fourier transforms

We have just reviewed a number of methods which CEOLAIRE'S melody extraction engine could use to generate frequency information, namely the DFT, FFT, STFT, filters and wavelets All can be used to achieve the same final goal, but the choice of which to use comes down speed and ease of use The discussion of which to use is left to Chapter 5 and until we have further explored issues relating to music and music information retrieval Now, we will move away from the digital signal processing topics of this dissertation and concentrate on what we can achieve now that we can access frequency information from audio files We start with a common tuning system for instruments called *Equal Temperament*

3 2 9 Equal Temperament

Western music is played by tuning instruments to a musical system known as **Equal Tempering** which evolved from the music theories of the ancient Greek mathematicians, although it was not until the 20^{th} century that it became the common tuning scale for pianos Equal Tempering is divided into octaves with 12 notes between each octave A piano standard, for example, has 88 notes across 8 octaves The Equal Tempering system assumes that the note known as middle A has a frequency of 440 Hz, two octaves down and the first note C starts at 32 70 Hz For the purpose of this dissertation, we limit the number of octaves in the Equal Temperament system to seven octaves, although more exist This is shown in Figure 3 20

Two corresponding notes in adjacent octaves differ in that one frequency is twice

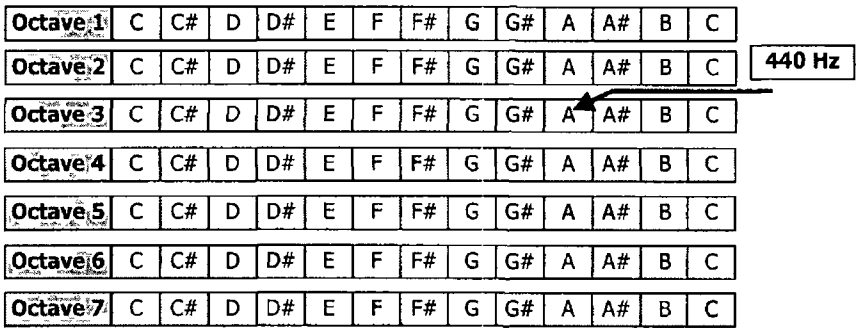


Figure 3 20 Notes and octaves

that of the other, i e they have a ratio of 2 1 The values that make up notes for the Equal Tempering system frequencies are calculated around a reference frequency using a difference of $\sqrt[12]{2}$ between frequencies Taking A=440 Hz as the reference frequency, the next note will be $\sqrt[12]{2} * 440 = 466\ 16$ Hz, the prior note is $\sqrt[12]{2} - 440 = 415\ 31$ Hz and so on The frequencies that make up the Equal Tempering system are shown in Table 3 8

Note	Octave 1	Octave 2	Octave 3	Octave 4	Octave 5	Octave 6	Octave 7
C	32 70	65 41	130 81	261 63	535 25	1046 50	2093 00
C#	34 65	69 30	138 59	277 18	554 36	1108 73	2217 46
D	36 71	73 42	146 83	293 66	587 33	1174 66	2349 32
D#	38 89	77 78	155 56	311 13	622 25	1244 51	2489 02
E	41 20	82 41	164 81	329 63	659 25	1318 51	2637 02
F	43 65	87 31	174 61	349 23	698 46	1396 91	2793 83
F#	46 25	92 50	185 00	369 99	739 99	1479 98	2959 96
G	49 00	98 00	196 00	391 99	783 99	1567 98	3135 96
G#	51 91	103 83	207 65	415 31	830 61	1661 22	3322 44
A	55 00	110 00	220 00	440 00	880 00	1760 00	3520 00
A#	58 27	116 54	233 08	466 16	932 33	1864 65	3729 31
B	61 74	123 47	246 94	493 88	987 77	1975 53	3951 07
C	65 41	130 81	261 63	523 25	1046 50	2093 00	4186 01

Table 3 8 Equal tempering frequencies
Reproduced from [Kientzle, 98, p18]

CEOLAIRE'S extraction engine is built around the values in Table 3 8 to help it identify notes We will see later (Section 5 2 1 2) how using Table 3 8, we can identify the occurrence of particular frequencies and hence musical notes Once we have defined which musical notes are present, the next question is "*How do we represent this information?*"

3.3 Representing Music and Melody

Music or the notes that make up the content of a musical song or melody can be represented by *Contour* or by *Interval* Contour is the representation of musical notes by the change or lack of change of contiguous notes over time Representing music or melody by Interval means representing musical notes based on the difference between adjacent notes

3 3 1 Representing Melody as Contour

Using Contour, the melody of a song can be defined by comparing the change from one note to the next over time These changes between adjacent notes and melodies are recorded as a note either going up (U), going down (D), or remaining the same (S) relative to the previous note, irrespective of any timing or duration information This language of three symbols is known as *Parsons* notation [McNab et al , 97] As the first note has no previous note, the Parsons notation starts with the change from the first note to the second For example, Figure 3 21 shows an array of notes and their Parsons equivalence

Notes	64, 65, 67, 72, 72, 67, 65, 64
Parsons	*, U, U, U, S, D, D, D

Figure 3 21 Representing musical notes using Parsons notation

When faced with the task of generating a musical query, which will itself have to be turned into some musical notation, a user will find it easier to generate a query using Parsons notation than having to recall exact note values Observing Figure 3 21, it should be clear that the cognitive load of generating a query using the exact note values is higher than trying to generate a query using Parsons notation if the user has to generate the query manually In the case of a query-by-example system, the cognitive load between the two methods may not be subject to the same kind of difference

Parsons is essentially a language with three symbols It can be further extended to a 5 symbol alphabet to allow for a stronger representation of music by defining

not only the note change, but qualifying that change based on how large a change occurred This is illustrated in Table 3 9

Symbol	Effect
-2	Large change downwards
-1	Small change downwards
0	No change
1	Small change downwards
2	Large change downwards

Table 3 9 Extended Parsons using five symbols

Using Parsons as a representation for music melodies incurs a loss in accuracy as the Parsons language only has 3 letters in its alphabet We will take a more detailed look at Parsons as a valid representation for music retrieval in Section 5 2 1 3 If we have access to music in a pitch like representation or where whole notes are identified by numbers, then it is a trivial task to convert it into Parsons All one has to do is observe the change from one note to the next

3 3 2 Representing Melody as Interval

With the interval representation, melody is defined by the difference between two adjacent notes in a melody Interval representation is shown in Figure 3 22

Notes	64, 65, 67, 72, 72, 67, 65, 64
Interval	+1, +2, +5, 0, -5, -2, -1

Figure 3 22 Representing musical notes using an Interval representation

As with Parsons, if we have a whole number representation, then it is a straight forward task to discover the Interval structure of a song The drawback with Interval is that a user is required to have a strong musical ability when generating musical queries We will observe the effects of using Interval as a representation for melody fragments in Section 5 2 1 3

3 3 3 Measuring Distance Between Musical Strings

We have seen that we can define music as a string, using the letters "U", "D" and "S" A string is defined as being a sequence of characters over a finite alphabet For example, the string "UDDDUSSUDDDU" is a string over the alphabet {U,D,S} The general problem of string matching is to locate all occurrences of a *target* string *t*, in a larger string *R* (drawn from the same alphabet), called the *reference* string Given three strings x, y and z, x is a *prefix* of the string xy, a *suffix* of the string yx, and a *factor* of yxz

In this section we review some of the more common methods for measuring occurrences and distances between strings, which could be the representation for melody using either contour or interval, starting with the methods used for exact string matching and how approximate string matching can be implemented to help rank strings in terms of their similarity to a query string

3 3 3 1 Exact String Matching

At its simplest, string matching can be undertaken using "brute force" Brute force string matching is shown in Figure 3 23 Here we can see the target string "abcacab" is being compared to a reference string "abcacaabcabcabcacabcab" The target string is laid out below the reference string and the characters in both are compared If a match is not found, the target string is moved across by one character and we compare again This "compare and move" process of matching character by character is undertaken continuously until a match is found

This is a computationally expensive process with a lot of unnecessary work being undertaken, requiring in the worst case, a total of m*n comparisons, where m is the number of characters in the reference string and n is the number of characters in the target string

Brute force matching requires a lot of comparisons which can be avoided and one way to cut down on the number of string comparisons is to match only when we have a potential match One such approach to achieve this is the Boyer-Moore algorithm

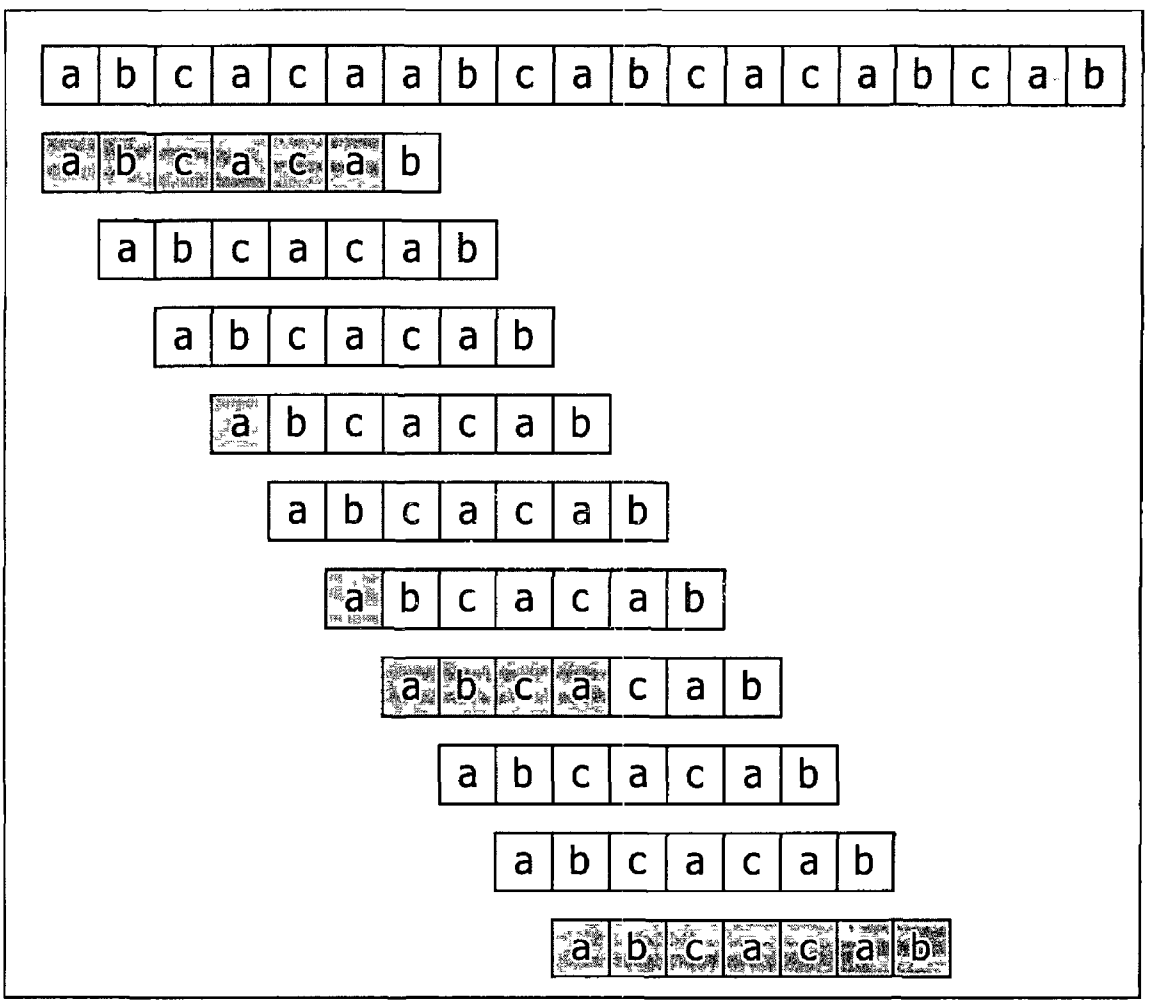


Figure 3 23 Brute force string matching

3 3 3 2 Boyer-Moore Algorithm

The Boyer-Moore algorithm [Boyer et al , 77] works by sliding a window containing the target string across the reference string using a *slide forward, compare backwards* strategy The Boyer-Moore algorithm slides forward through the reference string Within every window, a *backwards* search is undertaken, matching characters between the current location in the reference string and the target string The fact that we are starting at the end of the window and working backwards means that there is a potential reduction in the number of actual character comparisons which have to be undertaken For example, take the word "stapler" as the window and "stapled" as the target string The "r" and the "d" mismatch so it is known straight away after only one comparison that there is no need to continue comparing characters As long as the character in the reference string that mismatched does not appear anywhere in the target string, we can slide the window the length of the window forward in the reference string If the character that mismatched does occur inside the reference string we must shift the target string to the point in the reference string so that the mismatched character is aligned in both strings A backward comparison is then undertaken again This process is then continued until a match is found or the end of the reference string has been found An example implementation of the Boyer-Moore algorithm to find the target string "abcacab" in the reference string "abcacaabcabcacabcab" is shown in Figure 3 24

The steps to locate the string are as follows

1 A mismatch occurs straight away between "a" and "b" The mismatching character "a" does occur in the target string so we must align "a" with "a"

2 We start comparing backwards again and the mismatch occurs after the third comparison between "a" and "c" Again we forward align the target string to match "a" with "a"

3 Immediately, a mismatch occurs between "c" and "b" so we forward align the target string to align "c" with "c"

4 The next mismatch occurs after 4 comparisons between "b" and "a" Again,

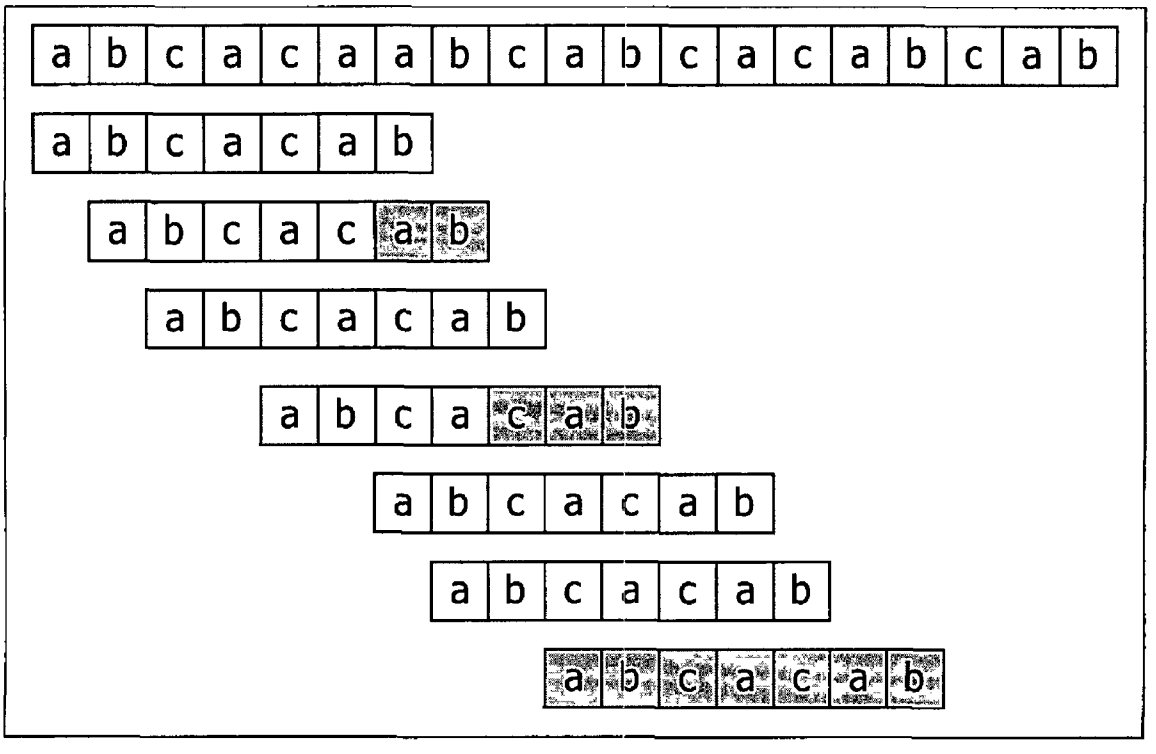


Figure 3 24 Boyer-Moore algorithm

forward alignment for "b" to "b"

5 Straight away, a mismatch occurs between "a" and "b" Forward alignment for "a" and "a"

6 Again, immediate mismatch occurs between "c" and "b" Forward alignment takes place for "c" and "c"

7 Match found!

Boyer-Moore has a worst case running time of $O(n+m)$ Another method for computing whether or not a target string occurs inside a reference string was discovered by Knuth, Morris and Pratt

3 3 3 3 Knuth-Morris-Pratt (KMP)

The Knuth-Morris-Pratt algorithm [Knuth et al , 77] works by sliding a window incrementaly through the reference string, comparing characters in a forward direction KMP employs a "failure function" which is used to decide where to position the target string against the reference string when a mismatch occurs

The purpose of the failure function is to compute the count of characters of the longest prefix of the target string that is a suffix within itself Basically, what is the largest sub-string starting from the 0^{th} character, that exists within the string The failure function is computed by comparing the target string against itself Every time a character matches, it is assigned an incremental cost of how many of the previous contiguous characters have matched The failure function for the string "abcacab" is shown in Table 3 10 It is computed as follows

1 Compare the 1^{st} character to the 0^{th} character $T[0] \neq T[1]$, so $ff(1) = 0$

2 Compare the 2^{nd} character to the 0^{th} character $T[0] \neq T[2]$, so $ff(2) = 0$

3 Compare the 3^{rd} character to the 0^{th} character $T[0] = T[3]$, so $ff(3) = 1$

4 Compare the 4^{th} character to the 1^{th} character $T[1] \neq T[4]$, so $ff(4) = 0$

5 Compare the 5^{th} character to the 0^{th} character $T[0] = T[5]$, so $ff(5) = 1$

6 Compare the 6^{th} character to the 1^{th} character $T[1] = T[6]$, so $ff(6) = 1 +$ the value of $ff(5)$ $ff(6) = 2$

j	0	1	2	3	4	5	6
T[j]	a	b	c	a	c	a	b
ff(j)	0	0	0	1	0	1	2

Table 3 10 KMP failure function

The failure function informs that if a mismatch occurs between the window and the reference string then the window can be shifted forwards to where the mismatch occurred, and then shifted back by the amount indicated in the failure function An example of the KMP algorithm is shown in Figure 3 25

KMP is implemented as follows

1 Start comparing forwards A mismatch occurs between the "a" and the "b" on the 6^{th} character of the target string The failure function is consulted, it states that a mismatch on the 6^{th} character incurs a shift of 2 The window is then shifted forward to where the mismatch occurred minus the amount from the failure function

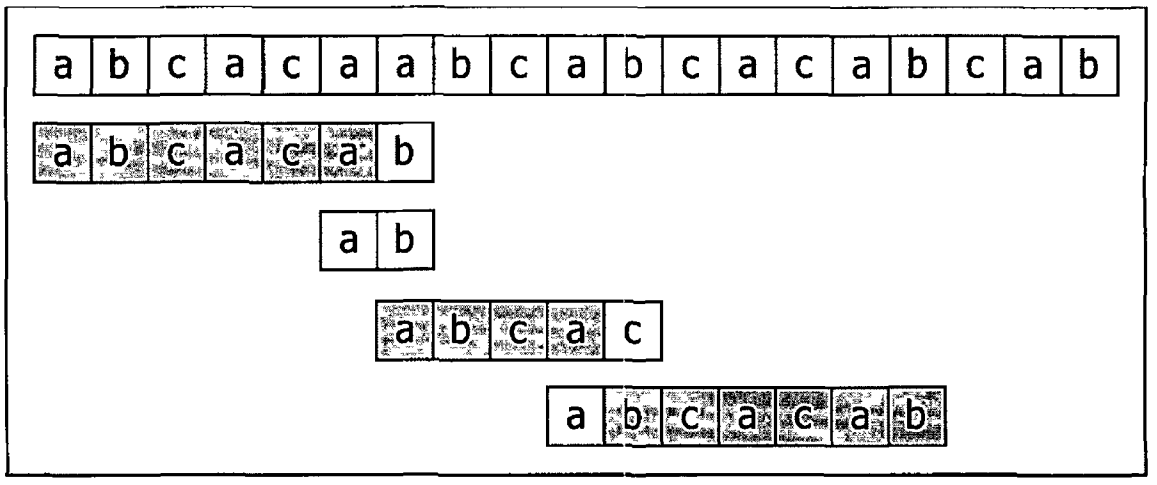


Figure 3 25 Knuth-Morris-Pratt algorithm

2 Now, we know that from the failure function there is no need to compare "a" and "a" We continue comparing forwards A mismatch occurs between the "a" and the "b" The failure function is consulted and has a shift cost of 0

3 The target string is aligned where the mismatch occurred, but this time we do not pull the target string backwards Then, after 5 comparisons a mismatch occurs between "b" and "c" Consulting the failure function, we observe a shift cost of 2

4 We compare forwards again, starting at "b" Match found!

The KMP algorithm has a worst case running time of $O(n + m)$

3 3 3 4 Bit-Parallelism

Bit-parallelism is a method which can be used to quickly compute the occurrence of a target string in a reference string if the length of the target string is smaller than the word size of the computer One of the early bit-parallelism methods is the Shift-Or algorithm [Baeza-Yates et al , 99b] The algorithm works with two word level bit-vector operations of computer hardware, SHIFT and OR The technique is called bit-parallelism since we are simulating a cost column in a single word Both the SHIFT and the OR operations are undertaken in *parallel* at the *bit* level The operations are implemented column by column from left to right until the entire

reference string has been processed or the bottom row of the vector contains a 0, indicating a match has been found

Shift-Or starts off by initialising a bit array showing the indices of all the characters in the target string The Bit-Array for the string "cacb" is shown in Figure 3 26

	a	b	c
c	1	1	0
a	0	1	1
c	1	1	0
b	1	0	1

Figure 3 26 Shift-Or bit array

The next step is to initialise a word sized vector E to all 1's , which is used in conjunction with the Bit-Array when applying the SHIFTs and ORs An example of running through the algorithm applying the Shift and Or operations is shown in Figure 3 27

	a			c			a			c			a			c			a			c			b			
c	1	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1
a	1	1	0	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	1	1
c	1	1	1	1	1	0	1	1	1	1	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	1	1
b	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0

Figure 3 27 Shift-Or table

Shift-Or starts by shifting the E vector by one (introducing a 0 at the top), then a vector from the Bit-Array corresponding to the character under observation is placed beside it The OR operator is applied to both, resulting in the next E column This process is repeated until the bottom row of the E vector is 0 or until there are no characters left in the reference string

This SHIFTing and ORing ensures that whenever a partial match is found a 0 is repeatedly passed on to the next E column and down by 1 Then when the final row of the E vector contains a 0 we know that we have a complete match

Shift-Or has a linear running time but as it uses hardware operations is very fast

The algorithms we have looked at so far have one thing in common They perform exact string matching, that is, they only find strings which are exactly the same A music information retrieval may want a certain level of "fuzziness" associated with it, exact string matching can be too limiting Take for example a query string "UUDDUSD" presented to a music information retrieval system There may not an exact match to that query string, but a similar string "UUDUUSD" may exist A pointer to the file containing this string could then be returned as part of a ranked list of similar files, ranked by the number of characters mis-matching This type of string matching is known as approximate string matching

3 3 3 5 Approximate String Matching

The goal of approximate string matching is to find the occurrences of a target string in a reference string, while the target and its occurrence in the reference string can have a limited number of differences The three popular approaches to approximate string matching are

- String matching with k mismatches
- String matching with k differences
- String matching with don't cares

String matching with k mismatches allows for matches where up to k characters in the target string do not match the reference string String matching with k differences requires that the target string has an edit-distance from the reference string with a value of k or less String matching with don't cares allow don't care characters to match any character The latter two matching functions are basically the problem of finding the longest-common-subsequence between the target string and the reference string Examples of the three approaches to approximate string matching are shown in Figure 3 28

3 3 3 6 K Mismatches

The problem of matching strings with k mismatches is to compare a target string against a reference string where at most k positions in both strings have differ-

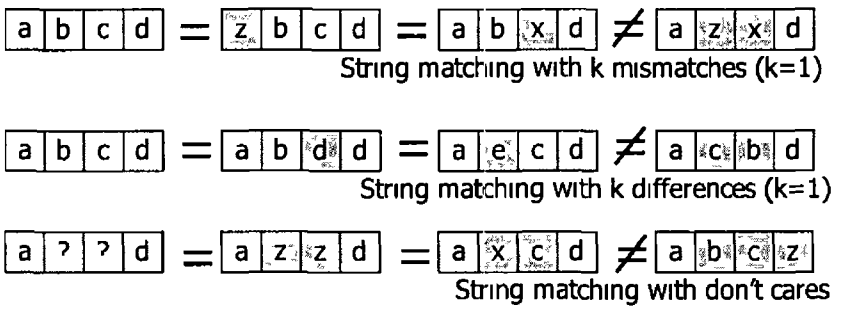


Figure 3 28 Different approaches to approximate string matching

ent characters k=2, would allow the strings "abcd" and "azzd" to be equivalent String matching with k differences does not allow for insertions or deletions of characters, only substitutions One method for computing the difference between two strings of equal length is to use a distance metric known as the **Hamming distance** [Hamming, 50], which is the number of positions with different characters When comparing two strings, the Hamming distance is the count of characters that differ between the two strings

Another way to compute whether two strings match with k mismatches is to use a version of the Baeza-Yates' Shift-or algorithm [Baeza-Yates et al , 99a]

3 3 3 7 K Differences And Don't Cares

String matching with k differences requires that the target string has an **edit-distance** or **Levenshtein Distance** [Levenshtein, 65] from the reference string with a value of k or less An edit-distance or edit-cost is the number of edit operations that can be performed to turn one string into another An edit operation can be the *insertion* of a character, the *deletion* of a character or the *substitution* of a character to another character The words "stable" and "staple" are similar and one can be morphed into the other with the *substitution* of just a single letter ("b" to "p") Further the word "staple" can be converted into the word "stale" with the *deletion* of a single character ("p") Conversely "stale" can be changed to "staple" with the *insertion* of a single character ("p") It is the assignment of costs to these editing mutations which gives an overall "distance" between the two strings being compared The edit-distance is defined as the minimum number of character mutations required to transform a target string into a reference string

The edit-distance uses dynamic programming to find an optimal solution to the minimum number of edits when turning one string into another Dynamic programing can for some problems reduce a problem with an exponential running time to a polynomial running time It does this by first breaking the problem into repeated sub-problems, finding the optimal solution for the sub-problems and then finding the overall optimal solution At the heart of the edit-distance algorithm is the use of a distance table

An example of finding the edit-distance between the strings "abcded" and "abddf" is shown in Figure 3 29 The algorithm works by first creating a matrix of size m*n, where m=size of "abcded" and n=size of "abddf" The first row is initialised with the values 0 m and the first column with values 0 n The table is processed column by column Starting with i=1, j=1, for every entry i, j (i=row, j=column), if the character represented by j is equal to the character represented by the row i, the cost is set to 0, if they are not equal the cost is 1 A value is then assigned to the distance matrix dist[i][j] using the minimum of

$$minimum \begin{cases} dist[i-1][j]+1 \\ dist[i][j-1]+1 \\ dist[i-1][j-1]+cost \end{cases}$$

Repeatedly calculating the distance values leads to the matrix shown in Figure 3 29 The actual distance is the number in the bottom right hand corner of the matrix, dist[m-1][n-1] In this case the edit-distance is 3 The morphing operations undertaken are two substitutions ("*c*" to "*d*" and "*e*" to "*f*") and one insertion (insert "*d*")

	a	b	c	d	e	d
a	0	1	2	3	4	5
b	1	0	1	2	3	4
d	2	1	1	1	2	2
d	3	2	2	1	2	2
f	4	3	3	2	2	**3**

Figure 3 29 Edit-distance of two strings

It should be clear how the edit distance can be used to undertake string matching with k differences It can also be used for string matching with k don't-cares as if we discover a don't care while calculating the distances, we simply treat any don't care character as a match

Figure 3 30 shows the calculation of the edit distance with 2 don't cares between the strings "abcded" and "ab??f" The matrix is calculated as earlier with the same rules, but every time we encounter a don't care symbol, we treat it as matching symbol, assigning a cost of 0 The edit distance between the strings "abcded" and "ab??f" is 2

	a	b	c	d	e	d
a	0	1	2	3	4	5
b	1	0	1	2	3	4
?	2	1	0	1	2	3
?	3	2	1	0	1	2
f	4	1	2	1	1	**2**

Figure 3 30 Edit distance with two don't cares

3.4 Summary

We started this chapter by exploring the differences between the time domain and frequency domain followed by an introduction to different methods of gaining access to musical content from digital music sources using frequency spectra generation and analysis We also explored some of the issues relating to their generation, for instance how signals have to be windowed before they can be reliably analysed We also took an in depth review of how Fourier transforms are derived and implemented Following that we looked at how music is played by tuning instruments to the equal tempering scale and how we can derive the values for that scale We also looked at how we can represent music by using both interval or contour and how music retrieval can be reduced to the task of string matching

Next, we will undertake a literature review which concentrates on some of the early work on music information retrieval and some of the systems that have been

implemented, both systems that work with MIDI and recent attempts to implement systems working on raw polyphonic musical sources

Chapter 4

Review of Previous Research in Music IR

4.1 Introduction

This chapter explores a number of different music information retrieval systems, from the early ones up to those currently under investigation by researchers We will see that MIDI as a format is still popular from a retrieval perspective, but that music information retrieval is moving towards allowing queries to be run against monophonic and polyphonic sources of raw or compressed audio signals rather than just MIDI tunes (which can be both monophonic or polyphonic) We will also cover score based systems such as Guido, and we will finish up the review by looking at some more commercialised approaches

Table 4 1 shows a summary comparison of some of the different music information retrieval systems which we review and how they compare to the system we built, CEOLAIRE

4.2 MELDEX

MELDEX is the New Zealand Digital Library Melody Index [McNab et al , 97] It was designed to retrieve melodies from a database on the basis of a few notes sung into a microphone A user uploads or gives MELDEX a URL of a file which contains

	User Interface	Index derived		Web based	Collection size	Music Type		Matching	
		MIDI	*RAW*			*Mono*	*Poly*	*Approx*	*Exact*
Arthur	N/A	No	Yes	No	100	No	Yes	Yes	N/A
MelodyHound	Whistling Graphical	Yes	No	Yes	10,000	Yes	No	Yes	Yes
ThemeFinder		Yes	No	Yes	N/A	Yes	No	Yes	Yes
OMRAS	N/A	No	Yes	No	N/A	Yes	Yes	N/A	N/A
QBH	Humming	Yes	No	No	183	Yes	No	Yes	Yes
Meldex	Singing Upload file	Yes	No	Yes	100,000 s	Yes	No	Yes	Yes
QPD	Graphical	Yes	No	Yes	183	Yes	No	Yes	Yes
SEMEX	N/A	Yes	No	No	N/A	Yes	Yes	No	Yes
Ceolaire	**Graphical**	**Yes**	**Yes**	**Yes**	**9,354 (NZDL)**	**Yes**	**No**	**Yes**	**Yes**

Table 4 1 Comparing some of the different MIR systems

a sampled acoustic signal Melody transcription is then performed by segmenting the acoustic stream identifying frequencies, applying pitch labels and note durations

For music transcription, only the fundamental frequency of the input is needed The input is filtered to remove as many harmonics as possible, while still leaving the fundamental frequency intact This is achieved by converting the input file's sampling rate to 22 kHz followed by its quantisation to an 8-bit linear representation Singing voice limits, ranging from F_2 (87 31 Hz) to G_5 (784 Hz), are used when analysing a query and any note outside the human singing range are not catered for

The acoustic stream is low-pass filtered with a cut-off frequency of 1,000 Hz, which has the effect of removing all frequency components above 1,000 Hz (including the harmonics) which aides in pitch discovery Pitch is discovered with the use of a pitch tracker, identifying pitches by the observation of repeating pitch periods which make up the overall waveform

Once pitches are identified, note start and end timings are determined MELDEX achieves this by depending on the user separating each note by singing either *da* or *ta* as this forces a short duration silence between notes as consonant sounds cause a drop in amplitude for 60 ms or more at each note boundary Adaptive thresholds can then be used to determine the note onsets and offsets Figure 4 1 shows the use of amplitude to segment a series of notes The rhythmic values are assigned by quantifying each note to the nearest minimum note duration, which is set by the user

MELDEX has a number of databases The Folksong database consists of 9,354

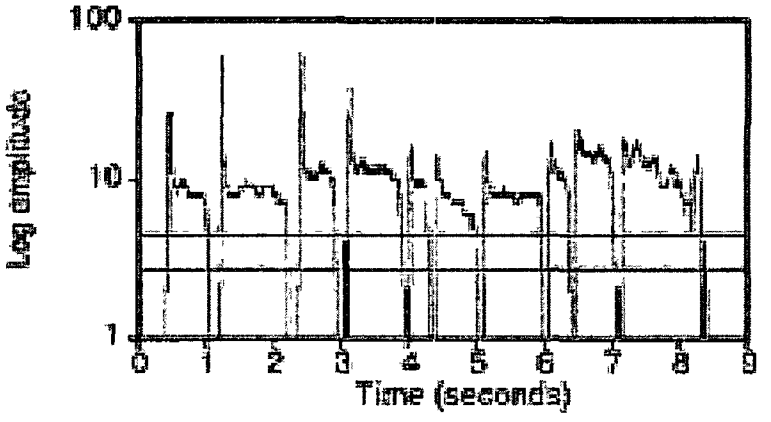


Figure 4 1 Note onsets and offsets of a query in the MELDEX system

Type 0 MIDI files taken from the following categories

- North American (and British) folk songs
- German ballads and folk songs
- Chinese ethnic and provincial songs
- Irish folk songs

MELDEX also has a Fake Book collection of approximately 1,200 popular tunes, a MidiMini collection and a MidiTheme collection which has over 17,000 MIDI files

A lot of modern songs contain a "hook", that memorable portion of the song that stays in your head Very commonly the hook is the chorus and therefore this is the portion of the song that a lot of people will want to search for MELDEX allows for search on either the explicit beginning of a song or at any location within a tune

MELDEX uses Parsons notation to build its index and this yields what is called the melodic contour or pitch profile A tune is effectively represented as a string from a vocabulary of 3 letters, thereby reducing the retrieval operation to the task of string matching Approximate rather than exact string matching algorithms are deployed in MELDEX as a query could easily have a Parsons symbol (a letter) out of place This type of error is introduced by the user *forgetting* a note, *inserting* a false note or *substituting* a wrong note in place of a correct one Hence the basic operations used in the string matching algorithms are *deletion* of a single symbol,

insertion of a single symbol and *substitution* of one symbol with another The three operations have associated costs which can be fixed or dependent on what the value of the current symbol is It is worth noting that these kinds of errors can not occur as the result of incorrect representations of melodies within the database, as the representations are taken directly from MIDI representations of the song

Some of MELDEX's advanced query features include note duration and tempo, search from the beginning of a song or anywhere in the song and interval rather contour matching MELDEX can also perform exact matching of interval and rhythm, interval regardless of rhythm, contour and rhythm, contour regardless of rhythm and approximate matching of interval and rhythm and contour and rhythm

4.3 Musclefish

Musclefish is a commercial application from Musclefish LLC which allows for content based retrieval of digital audio [Wold et al , 96] Musclefish achieves this by allowing access to sounds using the following properties

- *Simile* A simile is where sounds have some common characteristics, therefore they are similar Sounds can be classified in groups For example a sound can belong to the class of speech sounds or to the class of applause sounds

- *Acoustical / perceptual features* This involves describing the sound in terms of commonly understood physical characteristics such as brightness, pitch and loudness

- *Subjective features* This involves describing sounds using a personal descriptive language by training the system to understand the meaning of a particular term For example a user can train the system to know what a shimmering sound is

- *Onomatopoeia* Here, the system can retrieve a sound similar in some quality to the sound we are looking for For example, a buzzing noise could be presented as a query to find the sound of bees

The Musclefish audio database was built so that it could match and compare aural properties, not just melodies The system analyses sound to discover the following properties

- *Loudness* is approximated using the signal's root-mean-square (RMS) level in decibels This is calculated by breaking the sound up into a number of frames and computing the square root of the sum of the squares of the windowed sample values

- *Pitch* is determined by taking a series of short-time Fourier magnitude spectra The pitch is retrieved by using an approximate greatest common divisor algorithm on the frequencies and amplitudes of the peaks

- *Brightness* is computed as the centroid of the short-time Fourier magnitude spectra, stored as a log frequency Basically this is a measure of the higher frequency content of a signal

- *Bandwidth* is computed as the magnitude-weighted average of the differences between the spectral components and the centroid

- *Harmonicity* is the difference between harmonic spectra (vowels and most musical sounds), in-harmonic spectra (metallic sounds), and noise (spectra that seem to vary randomly in frequency and time) Harmonicity can be computed by measuring the deviation of the sound's line spectrum from a perfectly harmonic spectrum

Some of the applications that the Musclefish system can be used for include

- Audio database browser

- Search sounds for different properties

- Audio editors

- Allow sound editors a level of knowledge about the content of sound they are working with

- Surveillance
- Senses for an alarm system that listens for sounds, e g glass breaking
- Automatic segmentation of audio and video
- Detecting scene changes within video using audio

4.4 Query by Humming

The Query by Humming (QBH) system which was developed at Cornell University allows for musical information retrieval against a musical database [Ghias et al , 97] The architecture of the QBH system is shown in Figure 4 2

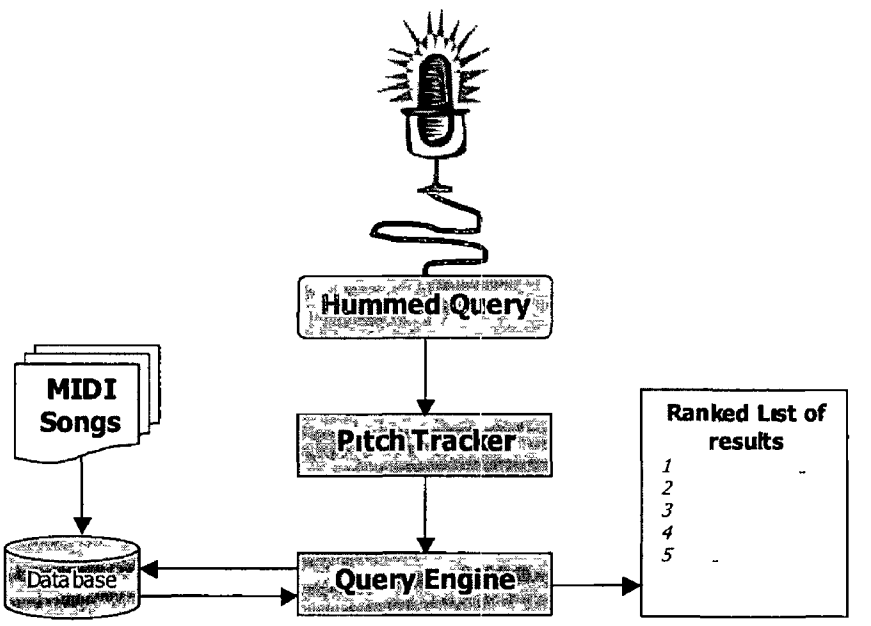


Figure 4 2 Architecture of the QBH system

QBH consists of 3 main components

1 Pitch tracking module

2 Melody database

3 Query engine

Queries are hummed before submission to the QBH system for analysis A pitch tracking module isolates and tracks the pitch of a user's hummed query using a

modified version of an auto correlation function The pitch tracking is implemented using Matlab, isolating and tracking peak energy levels of the signal resulting in an estimation of the pitch The query is then reduced to a string of Parsons notation

The musical database consists of only 183 MIDI files which are pre-processed into Parsons notation and stored separately in different files Matching a query to strings in the database is implemented with a degree of fuzziness The reason being to combat any errors which may be introduced by the way people hum and for any errors in the representation of the song itself The actual searching of the database is achieved using the Baeza-Yates approximate string matching algorithm, which can take transposition, drop-out and duplication errors into account

4.5 Query by Pitch Dynamics

Query by Pitch Dynamics (QPD), developed by The Link Group at Carnegie Mellon University [Beeferman, 97] is a system allowing for the automatic indexing of music by its tonal content so that it can be searched as well as analysed off-line QPD derives the tonal content of a song from a single discrete stream of pitch values which correspond to frequencies or notes from a musical file This one-dimensional time series sequence of elements is indexed based on the *relative* values of nearby elements, rather than *absolute* values These sequences are obtained from MIDI files

QPD stores note information in P-structures, where a P-structure is a pattern of notes from a tune An example P-structure from QPD is shown in Figure 4 3 The second note is the same as the first, the third is higher than the second, the fourth is lower than the third and so on It is basically a contour representation of a tune

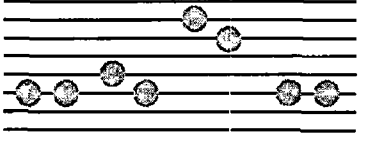


Figure 4 3 P-Structure for part of the tune "Happy Birthday"

It can be seen from the diagram that a P-structure shows only the relative difference between notes, not exact note values This takes some effort off the

user when formulating a query, as users are not expected to remember exact note sequences, only how the notes are approximately related

A crude estimate of the contour of a P-structure can be easily generated First the P-structure is divided into two parts, the left and right The sum of the ranks on the left hand side are compared with the sum on the right If the sum of the left has value greater than or equal to that of the right, a 0 is encoded, otherwise a 1 This is shown in Figure 4 4

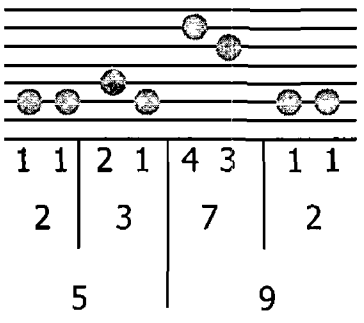


Figure 4 4 Estimating the contour of a P-Structure

The QPD system consists of a core engine which can run in one of four modes

- *Check Mode* simply scans all MIDI files looking for duplicates or corrupt files, which it discards It then produces the index
- In *Training Mode* the engine initialises the database and adds the index into the database
- *Query Mode* allows a user to perform exact and approximate matching on sequences in the database The sub-sequence recogniser can match songs based on a few successive notes In comparison with speech recognition software it can fail because notes can be inserted, deleted or substituted Sub-sequence queries are matched based on note relativity rather than exactness
- *Cluster Mode* searches the database for clusters This is achieved by analysing each window for its nearest neighbour

The system can be used to cluster songs that have similar melodic phrases or themes and to discover old or emerging genres of music It can also be used to identify different renderings of the same piece of music

QPD indexes a file using a sliding window to generate and record a P-structure for each window Each extracted P-structure is converted to a point in a feature space with a transform similar to that of the Haar wavelet transform [Flannery et al , 92] The points are then indexed in an R-tree and the Euclidean distance is used as the distance metric at query time

4.6 ThemeFinder

ThemeFinder [Kornstadt, 98] is a web-based melody search system [Themefinder, 02] allowing searching of music using a variety of methods Music can be searched using pitch, interval, scale degree (do, re, mi etc), contour and refined contour, either from the beginning only or from anywhere within the song Searches can also be key dependent in either a major or minor mode or both The user interface for the ThemeFmder system is shown in Figure 4 5 ThemeFinder searches against a database of classical, folk song or Renaissance tunes

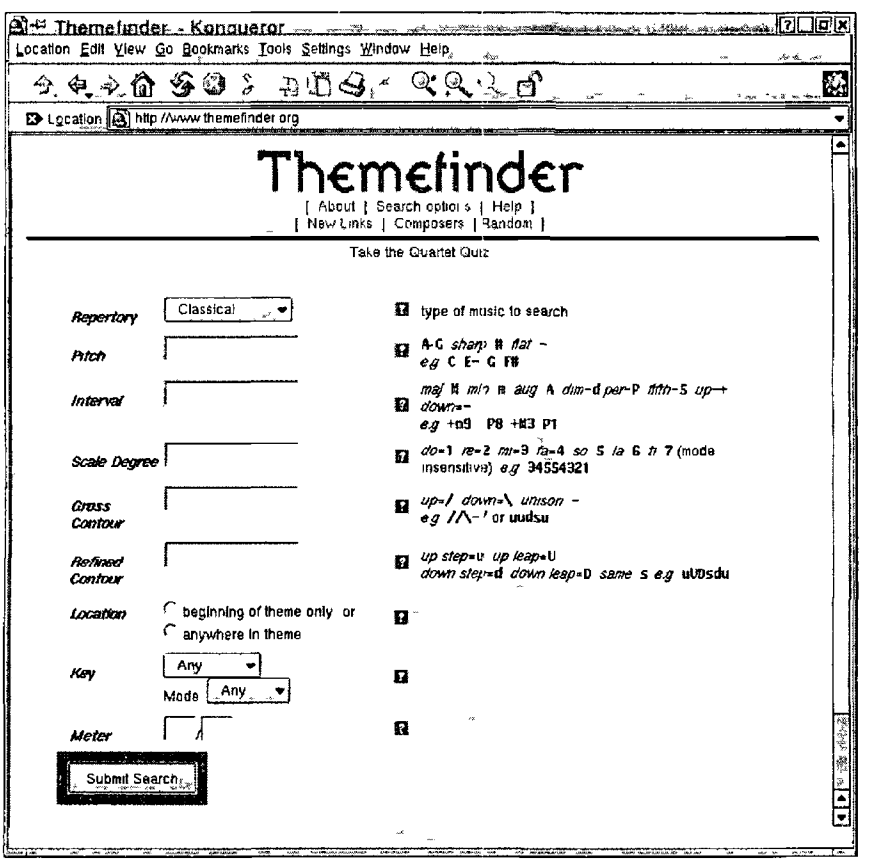


Figure 4 5 The user interface for the ThemeFinder system

4.7 MelodyHound

MelodyHound [Melodyhound, 02] is a web-based music information retrieval system which allows users to retrieve songs by either entering Parsons notation directly or by whistling a query A user downloads a Java applet which allows the recording of a whistled tune This tune is then analysed by the applet for frequency information and Parsons notation is automatically derived from this frequency information The Parsons notation is then submitted to the melodyhound database which consists of about 10,000 classical songs, 100 popular songs, 10 folk songs and 100 national anthems When the answer set is returned, the user is presented with a link to purchase the music at an online music store or to purchase the score of the song The MelodyHound interface is shown in Figure 4 6

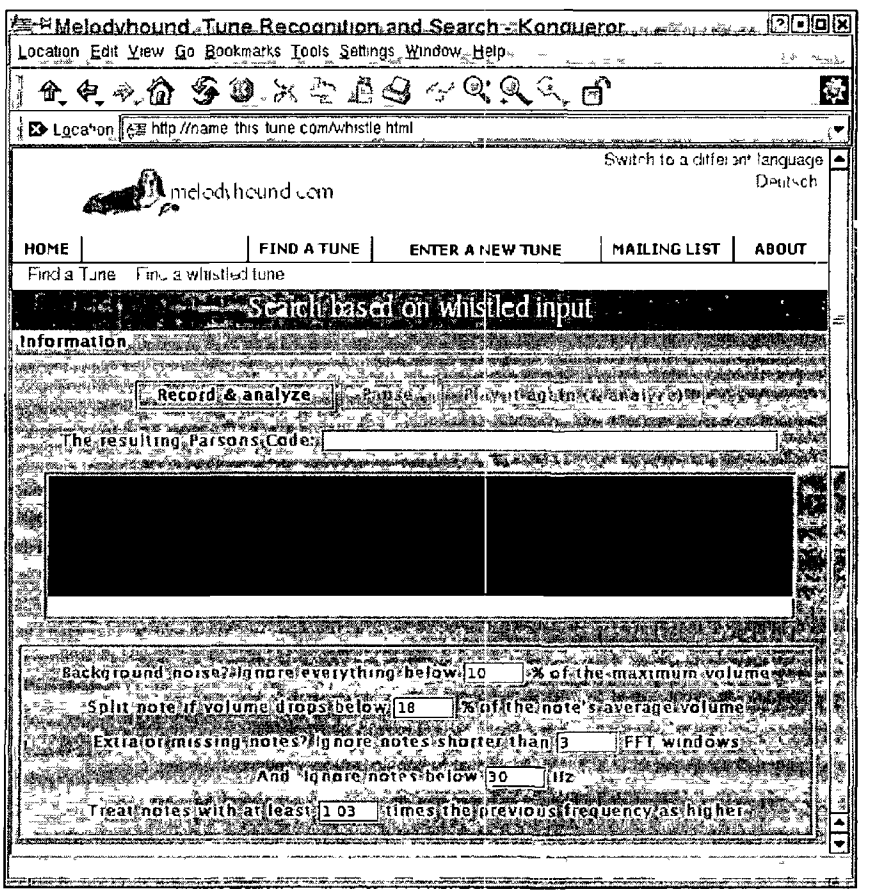


Figure 4 6 The user interface for the MelodyHound system

4.8 ARTHUR

ARTHUR [Foote, 00] is an experimental audio retrieval-by-example system designed for orchestral music It is named after Arthur G Lintgen who was able to identify phonographic recordings by the softer and louder passages, which are visible in the grooves of an LP ARTHUR was designed to do the same, retrieve audio based on its long term structure The experiments carried out on ARTHUR were done using a number of different classical CDs Audio energy versus time in one or a number of frequency bands is one way to determine the long-term structure of audio At the core of ARTHUR is the use of a dynamic programming algorithm called "Discrete Time Warping", which is used in speech recognition to help account for variations in speech timing and pronunciation Dynamic Programming is useful where signal amplitudes do not match exactly and relative timing is divergent, as is the case when two different orchestras perform the same piece

Two signals are aligned to each other via a lattice, the test signal on the vertical axis and the reference on the horizontal Every point (i, j) in the lattice corresponds to how well the reference signal at time i corresponds to the test signal at time j The Dynamic Programming algorithm returns the best aligned path that converts one signal into the other and the cost of that path If the 2 signals are identical, the cost will be 0 and the resulting line will be diagonal Increasing dissimilarity will have increased costs associated with the best aligned path

[Foote, 99b] also has an excellent review of audio information retrieval with a section dedicated to music information retrieval

4.9 SEMEX

SEMEX [Lemstrom et al , 00] is an efficient music retrieval prototype which uses bit-parallel algorithms for locating transposition invariant monophonic queries within monophonic or polyphonic musical databases SEMEX uses pitch levels for music representation Pitch levels are represented as small integers, making up the language of the system The architecture of the SEMEX system is shown in Figure 4 7

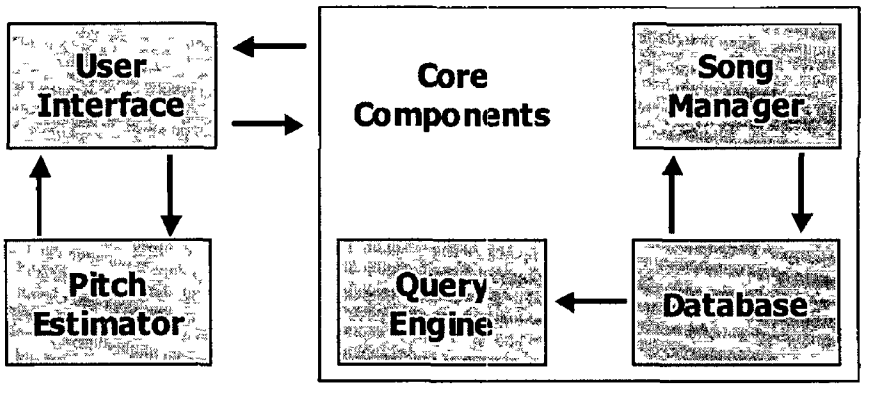


Figure 4 7 Architecture of the SEMEX system

If the database is comprised of monophonic music representations, SEMEX applies an adaptation of Myers algorithm [Myers, 98] Myers algorithm is a fast bit-vector algorithm for approximate string matching based on dynamic programming The matching algorithm was modified so that it only reports positions where the edit distance does not grow and positions which are not followed by a better match This is known as pruning The pruned results must first be verified before the results are returned to the user This is because Myers algorithm only computes edit distances The matching position j is passed to the dynamic programming algorithm to resolve a match

Retrieval of polyphonic sources in SEMEX can be accomplished using one of two methods First, the music database is reduced to a monophonic one Poly to monophonic reduction is achieved by simply selecting the highest notes of the chords Retrieval can then be performed using the monophonic techniques Alternatively, SEMEX can leave the database in its polyphonic form and search for transposed exact occurrences using its MonoPoly algorithm The MonoPoly algorithm is a two-phase algorithm The first phase pre-processes the source and the second phase applies a filtering technique and consists of two sub-phases The first of these sub-phases is the marking phase which performs bit-parallel searches for possible occurrences of a given query pattern These possible occurrences, when found, are called *candidates* The second of the two sub-phases, the checking phase, then scans through the candidates looking for actual occurrences The speed of the MonoPoly algorithm is heavily dependent on the output of the marking phase SEMEX uses MIDI files with an average polyphony degree varying between 1 and

3 The maximum polyphony degree observed was 41

4.10 OMRAS

Online Music-Recognition and Searching (OMRAS) is a cross-disciplinary research project covering computational musicology, computer science and library science to answer the problem of a digital music library, the inability to search the content of the collections for music itself One of the goals of OMRAS is to offer access to and retrieval of polyphonic sources with recent work by [Plumbley, 01] allowing access to raw polyphonic music sources and a framework for performing retrieval on polyphonic sources [Dovey, 01]

4.11 Sheet Music Searching

The systems we have looked at so far have mostly been concerned with retrieving music using contour or interval We may also want to search music in sheet form which exists on a musical staff

One system which allows music to be searched for in this form is Guido [GUIDO, 02] Guido [Hoos et al , 01] is a music information retrieval system which supports the following types of queries

- Absolute pitch (c1, d#2, etc)
- Intervals (minor third, major sixth, etc)
- Interval Types (second, fourth, etc)
- Interval classes (equal, small, medium, large)
- Melodic trend (upwards, downwards, static)
- Absolute durations (1/4, 1/8 , etc)
- Relative durations (1 2, 4 3, etc)
- Rhythmic trends (shorter, longer, equal)

Guido uses its own language to represent music Gudio performs retrieval using a probabilistic model (Hidden Markov Model) It can also be used to generate graphics of music on a musical staff In Figure 4 8 we can observe the Guido Java applet being employed to create sheet notation representing a query

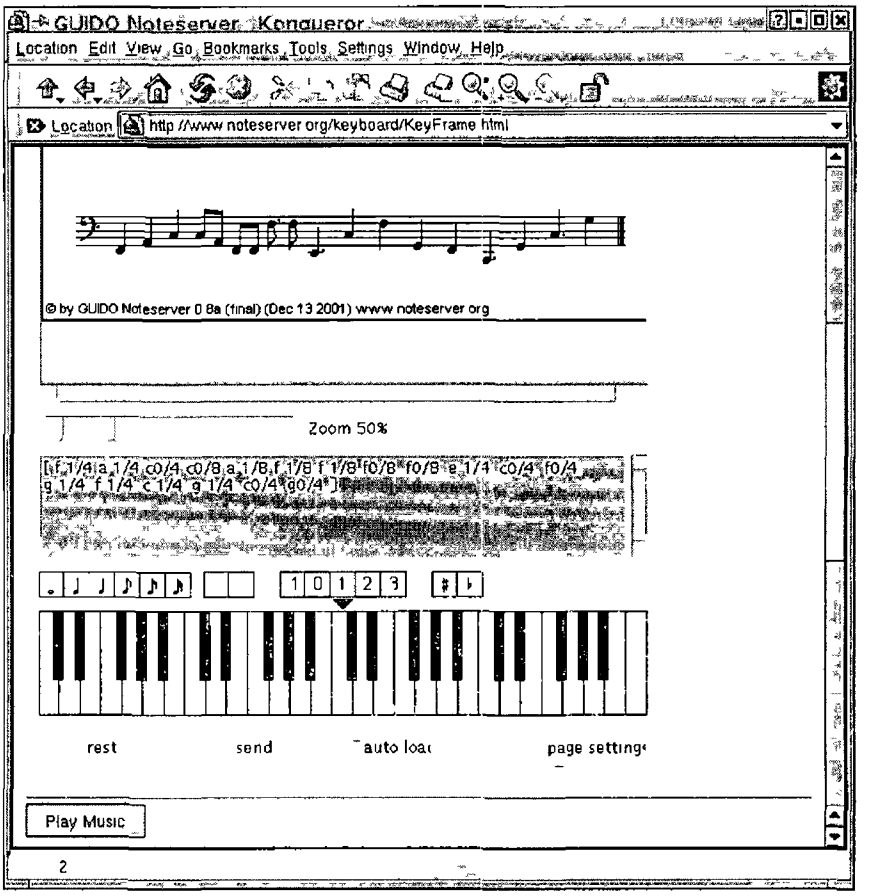


Figure 4 8 Gudio note composer

4.12 Prominent Search Engines

Prominent search engine companies like Altavista [Altavista, 02] and FAST Search and Transfer ASA [Alltheweb, 02] provide text based searching of musical metadata Examples of metadata searching include

- Band, Artist, Title, Album,
- File type
- File size

- Duration

Other recent advances in music information retrieval include Philips audio fingerprinting system [Philips, 01] which creates a digital fingerprint of a song so that when a user presents a song to the system, the song is identified and the user is presented with the option to buy the album containing the song Digital fingerprinting is achieved by breaking the song into segments of 10 milliseconds and then measuring the energy of 33 frequency bands of each segment Using a special algorithm, the energies are transformed into a code, which is unique to each song Similarly, [Starcd, 02] has a system which continuously listens to and identifies songs playing on a number of popular American radio stations, allowing a user to purchase music that is currently playing on a particular station or to browse through an archive of time stamped songs

4.13 Napster

The Napster [Napster, 02] online music service was founded in May of 1999 as a peer-to-peer file sharing service, allowing users to easily trade music encoded in the MP3 format MP3 was growing in popularity and Diamond Multimedia had already introduced their portable MP3 player This meant that people could now not just store their MP3's on their computers but they could carry them on their person

Napster was to become a thorn in the side of the recording industry as it allowed users to trade their music files over the Internet The problem that Napster faced was that their service was being used to share copyright material and as a result they were eventually forced under law to shut down their file swapping services Currently Napster is reviewing its technologies to re launch as a fee based service

4.14 Summary

In this chapter we reviewed the early work on music information retrieval and some of the systems that have been implemented, both systems that work with MIDI and recent attempts to implement systems working on raw polyphonic musical sources

What can be observed from this literature review is that the current trend is towards building systems which can successfully negotiate features or structures from polyphonic sources In the last few years the field of music information retrieval has been extended by incorporating the use of raw or compressed music files as a basis for experimentation [Foote, 00] [Plumbley, 01], although the use of MIDI as a musical format for music information retrieval is of particular use in testing proof of concepts for both polyphonic [Doraisamy, 01] and monophonic music sources as its structure is easily extracted

This chapter finished up our discussion and background information for music retrieval In the next chapter we will take a look at CEOLAIRE, the music information retrieval system we built based on the principles we have observed so far and the system upon which this dissertation is based

Chapter 5

CEOLAIRE

5.1 Introduction

In this chapter we introduce and review the CEOLAIRE music information retrieval system, which we built to implement the ideas on music information retrieval described in this dissertation The system can be described in terms of three interrelated components

- Music File Processing
- Search Engine
- User Interface

Figure 5 1 shows the core components of CEOLAIRE The preliminary stage of building an information retrieval system is to gather documents for indexing Once music files have been gathered, they are processed, turning raw music files into "documents" suitable for indexing The second stage takes the documents that have been processed and submits them to the indexing queue and builds a music index The search engine consists of a query manager which undertakes retrieval using the *tf*idf* weighting scheme incorporating document length normalisation The third component is the user interface consisting of a Java applet allowing users to generate, view and review music queries, which are in essence music fragments, before they are submitted for retrieval

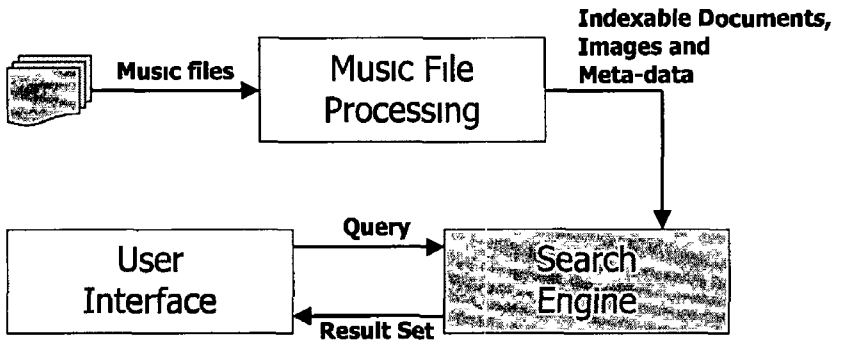


Figure 5 1 The core components of CEOLAIRE

5.2 Music File Processing

The goals of music file processing are threefold It has to

1 Generate a png format graphic of the music file (for presentation in the answer set)

2 Generate valid indexable documents of music fragments consisting of overlap ping n-grams

3 Create a record for this music file in the meta-data database

Figure 5 2 illustrates the components of the Music File Processing stage, which are achieved using two important components, the first being *melody extraction*, the second being *meta-data generation*

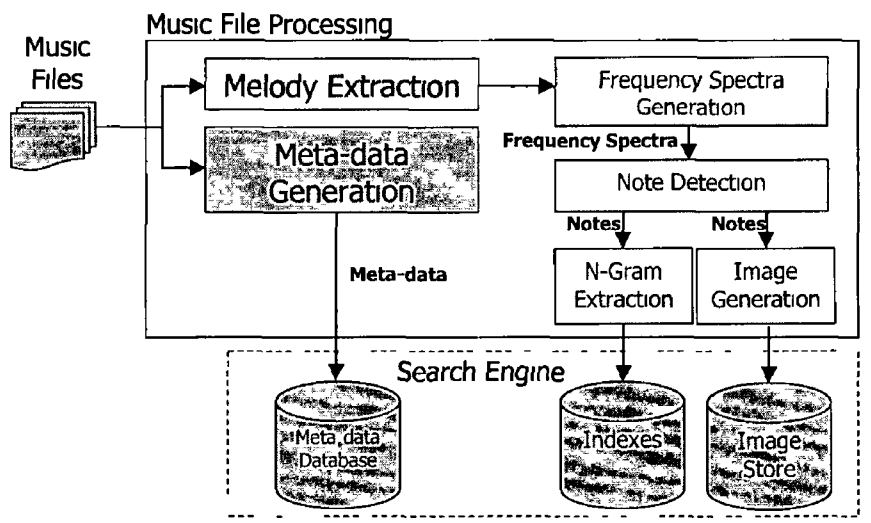


Figure 5 2 Music file processing

5 2 1 Melody Extraction

Melody is extracted from the music files using a two stage process The first generates frequency spectra while the second takes a frequency spectrum and identifies notes on the Equal Tempering scale Equal Tempering filtering was covered in section 3 2 9

5 2 1 1 Frequency Spectra Generation

In section 3 2 we discussed a number of different methods which could be used to gain access to the frequency information of a PCM encoded music file There are a number of approaches which can be used to achieve this goal We could use the Discrete Fourier Transform to generate spectra, but the processing time would be excessive As we saw earlier, the running time would be in the order of $O(n^2)$ Another approach available to us is to use a series of band-pass filters, to filter out notes with their corresponding harmonics, but this would require the generation of filters for each note on the Equal Tempering scale We could also have used wavelets, but in the end our preferred method for a number of reasons was to use the fast Fourier transform (FFT) as frequency spectra are easily generated using the FFT with a running time of $O(nlogn)$ This allows us to generate detailed frequency spectra quickly, to which we are able to apply a note filtering function Figure 5 3 illustrates the steps required to generate frequency spectra They are

1 Window samples

2 Apply Hamming function to windowed samples

3 Apply a 32,768 point FFT to windowed samples

4 Increment window reference in music file and repeat

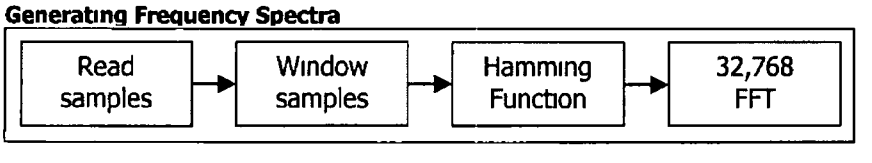


Figure 5 3 Generating frequency spectra

Music files are processed incrementally with a non-overlapping sliding window The size of each window is set so as to allow for the accommodation of at least 20 processed windows per second Each window is moved forward by 0 05 seconds In the case of a music file with a sampling rate of 32,000 Hz, the window size is 1,600 samples Stereo channels are averaged into one channel

A Hamming function is applied to each window to converge the sampled signal towards zero at both end points, reducing any sharp jumps that may be present, as they would result in the introduction of noise This processed window is then passed to a 32,768 point FFT function, generating a frequency spectrum for the given window We implemented a 32,768 point FFT function as this enables us to generate fine-grained spectra with a minimum resolution of 1 3 Hz Frequency resolution is calculated by dividing the sampling rate by the size of the FFT

We have already seen that the frequency spectrum generated as a result of a FFT is dependent on two things in particular, the sampling rate and the size of the FFT For example, if the sampling rate is 44,100 Hz and the FFT is a 32,768 point FFT then the spectrum is divided up into frequency ranges of 1 3 Hz In the case of a music file sampled at a rate of 32,000 Hz with a 32,768 point FFT, the resulting spectrum is divided up into frequency ranges with each range represented in steps of 0 9 Hz It can be observed that the lower the sampling rate with a constant sized number of FFT points, the more informative the frequency spectrum

1 3 Hz is the smallest acceptable increment that can be used with our Equal Tempering filtering algorithm The difference between C and C# in the first octave (as used within CEOLAIRE) is 1 95 Hz, Note C being 32 70 Hz and Note C# being 34 65 Hz We would simply not be able to reliably analyse notes in this octave without using this many points for the FFT We don't believe there is a need to go any lower than 32 70 Hz as most music is played in the octaves around the reference frequency However as we will see in the next section, we are not interested in the entire spectrum, just a portion of it The spectrum is then passed to the Equal Tempering filtering function

5 2 1 2 Note Extraction

This stage of the melody extraction process filters out Equal Tempering notes The algorithm can be adapted around other music scales with different reference frequencies and for microtonal scales such as are found in Indian music which has twenty-two notes in an octave Western music has twelve notes in each octave, with the interval between each note called a semi-tone Currently CEOLAIRE is limited to analysing Equal Tempering notes, but the principles that our extraction engine is built upon can be extended to work for other musical scales, Indian music for example Analysing the frequency spectrum of a sound created by an instrument shows a strong presence of a *fundamental frequency* (the note being played) followed by a decaying amplitude presence at integer multiples of the fundamental frequency

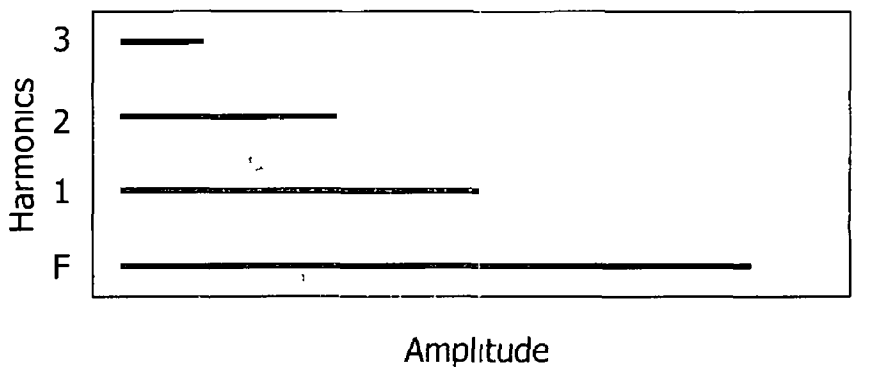


Figure 5 4 Fundamental frequency with harmonics

This fact is the basis of CEOLAIRE'S extraction engine If we know that a note has a pre-defined amplitude presence in the frequency domain when it is played, then we can analyse the frequency domain for this occurrence CEOLAIRE'S extraction engine exploits this fact for instruments tuned to this temperament, although not all instruments have harmonics at exact multiple intervals of the fundamental frequency Sounds created by a piano, for example, can be subject to the phenomenon known as *stretched partials* In chapters 6 and 7 we present results which show how this process which can be subject to errors effects extraction and retrieval effectiveness

Equal tempering filtering is a two stage process First, a two dimensional note discrimination table is dynamically built This table is dependent on the sampling rate and the size of the FFT employed Each column in the table corresponds to

Fundamental Frequency	32 70 Hz
Harmonic 1	65 41 Hz
Harmonic 2	98 10 Hz
Harmonic 3	130 81 Hz
Harmonic 4	163 50 Hz
Harmonic 5	196 20 Hz
Harmonic 6	228 90 Hz

Table 5 1 The first six harmonics of note C (Octave 1) as played on a piano

a particular note on the equal tempering scale, while each row corresponds to the presence of either a fundamental frequency or one of its harmonics

This is the basis of the equal tempering filtering algorithm we employ and is illustrated in Figure 5 4 The figure shows that there is a fundamental frequency which has a strong amplitude presence, with decreasing amplitude presence at each of the harmonics Rather than just picking out the strongest frequency component, the filtering algorithm explicitly searches for the presence of a fundamental frequency with its corresponding harmonics When filtering out notes in the presence of other sounds or notes in a sample, this method may prove to be more robust Currently the algorithm limits the note detection range to 72 notes over 7 octaves, but this can easily be extended to include more notes at a cost which is measured in processing time, although this would be unlikely to prove useful This note range is adequate for the dataset which we employ as all the notes fall within the 72 note range We will go into a lot more detail about the dataset in Section 5 2 1 3 The filtering table is comprised of a column representing each note Each row in the column corresponds to a particular frequency from 0 to half the sampling rate and contains a weighted value for the presence or lack of presence of a note Figure 5 5 shows the structure of the table

The pseudocode algorithm for building the Equal Tempering filtering table is shown as Algorithm 1

We know that the Equal Tempering scale is defined by Middle A = 440Hz This means that the frequency of the first note C can be calculated It is equal to 32 70 Hz To allow for minor imperfections in tuning and stretched partials, the algorithm sets a lower and upper bound where the amplitude presence should lie, rather than

Algorithm 1 Building the note discriminating table

```
-- Variable Declaration
MAX_NOTES   int {constant}
MAX_STEPS   int {constant}
note                                    int
step                                    int
harmonic                                int
upper_note                              int
lower_note                              int
noteRefTable[MAX_NOTES][MAX_STEPS]      float
-- Variable Initialisation
upper_note = 33 658183138
lower_note = 31 769094476
note_inc   =  1 059463094
-- Algorithm
for note = 0 to MAX_NOTES, increment by 1
 for harmonic = 1 to 10, increment by 1
  upper_limit = upper_note*harmonic
  lower_limit = lower_limit*harmonic
  for step = 0 to MAX_STEPS, increment by 1
   test=step*hz_per_step
   if test >lower AND test < upper then
    noteRefTable[step][harmonic] =1 0/(step*harmonic)
   endif
  endfor
 endfor
 upper_note = upper_note*note_increment
 lower_note = lower_note*note_increment
endfor
```

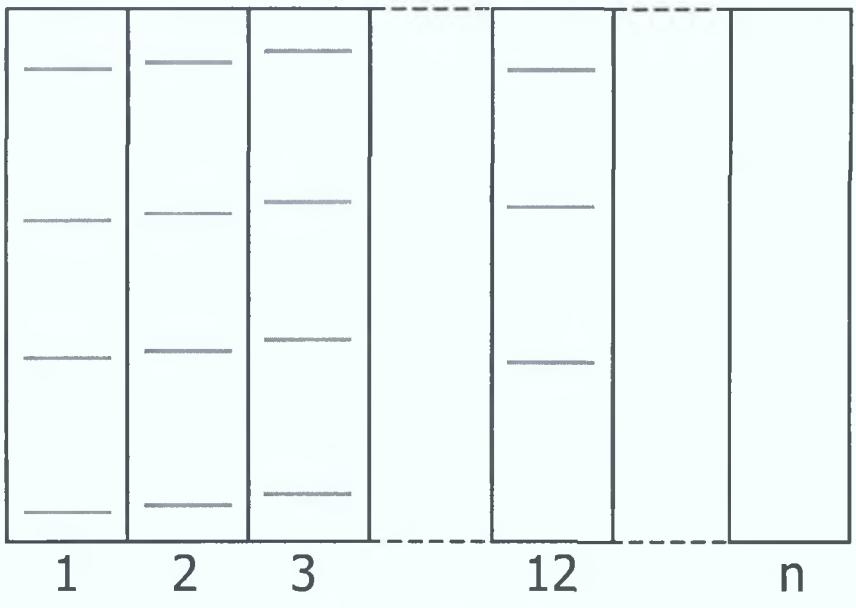


Figure 5.5: Equal tempering filtering table

explicitly assigning weights at the predefined points. It can be observed that note C in Algorithm 1 is defined by a lower bound limit of 31.77 Hz and an upper bound limit of 33.66 Hz.

The algorithm starts by assigning a weight for the fundamental of the first note C, followed by decreasing weights for each harmonic. It repeatedly does this for each note. The table is constructed each time a file is processed as each row represents a frequency which is dependent on the size of the FFT and the sampling rate. In the case of a music file sampled at 44,100 Hz and employing a 32,768 point FFT, each row represents steps of 1.3 Hz. Similarly, if the music file is sampled at 32,000 Hz then each row represents steps of 0.9 Hz.

The frequency spectrum from the FFT function is passed to the note discriminating filter. Both the filtering table and the frequency spectrum are stored as arrays. The filter function takes an empty array and the FFT'd samples as parameters. The empty array, called **notes_present**, holds the relative amplitude of any note which may be discovered. Each row in this array corresponds to a note on the equal tempering scale. Each entry in the **notes_present** array contains the sum of the multiplication of the FFT'd samples by the array corresponding to current note. This is shown in Figure 5.6. If the overall result of this multiplication and addition is positive, then the note is considered to be present. The pseudocode for

the notes filtering algorithm is shown as Algorithm 2

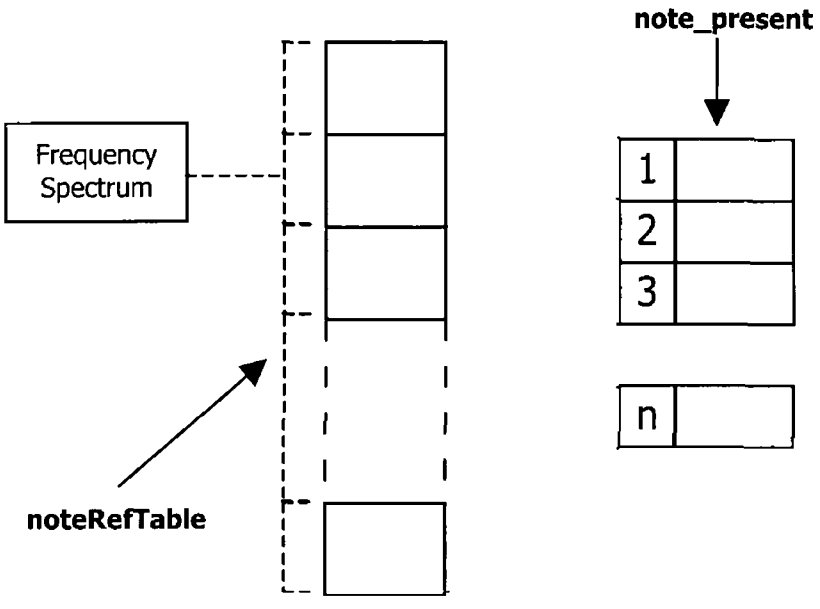


Figure 5 6 Note detection

The output of the note filtering algorithm is an array containing the relative amplitude of any notes which are present The next step of the process is to analyse the notes for contour and interval before generating indexable documents

5 2 1 3 Document Generation

Central to any information retrieval system is the use of a corpus of documents A web search engine for example uses HTML documents, a video retrieval system video clips, an image retrieval system uses image files and a music retrieval system utilises music files The initial raw or compressed files are usually processed to develop structures which describe content upon which we perform retrieval

The corpus of music files which we employ for our experiments are taken from the NZDL folk-song collection [McNab et al , 97] This collection of files consists of 9,354 music files from a mix of American, Chinese and European folk-songs which makes it representative of music from various cultures and the collection has been processed by McNab so that it does not contain any duplicate songs The collection is in the MIDI format which means that it is easily transferred as the entire collection is only 38 MB We calculated the total amount of music to be in excess of 62 hours

Algorithm 2 Algorithm to determine if a note is present

```
-- Parameters
MAX_NOTES                                  int {constant}
MAX_SAMPLES                                int {constant}
result[MAX_NOTES]                          float
samples[MAX_SAMPLES]                       int

-- Variable Declaration
MAX_NOTES                                  int {constant}
MAX_STEPS                                  int {constant}
note                                       int
step                                       int
noteRefTable[MAX_NOTES][MAX_STEPS]         float

-- Algorithm
for note = 0 to MAX_NOTES, increment by 1
 result[note] = 0
 for step = 0 to MAX_STEPS, increment by 1
  result[note] = result[note] +
                 (fft_samples[step] * noteRefTable[note][step])
 endfor
endfor
```

Ceolaire is sensitive only to western music or more explicitly music played using equal tempering Traditional Indian music is not played using equal tempering and therefore falls outside the the equal tempering filtering algorithm Arab and Chinese music also have their own scales which developed over time We view the extraction engine as a basic model which can be changed to adapt to other scales and instruments

CEOLAIRE uses the *tf*idf* weighting scheme incorporating document length normalisation to weight terms for retrieval *tf*idf* has been used with success by text based information retrieval systems In such systems each term in the corpus represents a word and every term has semantic content In some cases, this content can be verified by the use of a dictionary or thesaurus If its semantic context cannot be verified through a source the term may be discarded An example of a term with no semantic content is a misspelled word Terms in CEOLAIRE do not have semantic content, there is no dictionary which can be consulted, rather they are simply sub-string melody representations of an overall melody We have seen how CEOLAIRE derives this contour during the pre-processing stage The sub-strings are

melody fragments When *tf*idf* was developed, it was assumed that the more frequently a particular term appears in a document the more likely that term is a good descriptor of the document It is worth noting that while the term frequency (tf) part of *tf*idf* can clearly be used for indexing music sub-strings, the use of idf is not as clear-cut The frequency of a terms occurrence in a text document can be used indicate that terms significance to the content of the document Terms with a high frequency of occurrence are considered to be too common while terms with a low frequency occurrence are considered to be too rare to contribute to the retrieval process The terms which do contribute lie between the two extremes, the mid-frequency terms While it is clear that this applies to English language documents, it is not clear whether this applies to music documents composed of sub-strings Terms which appear across many documents will influence the ranking process less than rarely occurring terms, which is a consequence of using idf We use *tf*idf* to index music documents, however we are aware that it was designed for use with English language text, not music sub-strings

A indexable document to CEOLAIRE is a document containing music fragments represented using either Parsons or Interval notation Both Parsons and Interval notation are abstract descriptions of a music file CEOLAIRE segments this notation into sub-strings, effectively creating sub-strings of music fragments We accept that the terms do not have semantic content when generating the *tf*idf* weights for them Based on the theory behind *tf*idf* for text retrieval, we assume that the more frequently a melody fragment occurs in a music file, the better that melody fragment describes the music file In essence, what we are providing is a form of melody fragment searching within a corpus consisting of documents of melody fragments

The Document Generation stage is divided into two similar processes The first is to generate indexable documents using Parsons, the second using Interval Parsons notation is used for simple searching where a user has limited knowledge about music while Interval notation is for more experienced users who feel they have a intimate knowledge of music For both, the input into the Document Generation process is the **note_present** array for each of the time slices in the music file under investigation Currently CEOLAIRE utilises a monophonic music corpus so it is a relatively straightforward task to define the music contour The note_present arrays

are analysed consecutively When a note change is detected it is recorded in Parsons notation as up (U), down (D) or same (S) relative to the previous note This leads to a single string of Parsons notation Similarly, for interval derived documents the difference between note changes is recorded, generating a string of note changes The next step is to convert the abstract melody fragments into n-grams

N-grams have found many uses in the field of information retrieval, from cryptography [Pratt et al , 42] to stemming [Adamson et al , 84] to language identification [Damashek, 95] and so on N-grams can also be used to identify spelling errors and attempt to correct these within a information retrieval system [Angell et al , 83] [McIllroy, 82] [Morris et al , 75] [Peterson, 80] etc

To generate n-grams of a term, that term is segmented into smaller overlapping terms of fixed length n This is shown in Table 5 2

n-grams	n
m a r m a l a d e	n=1
ma ar rm ma al la ad de	n=2
mar arm rma mal ala lad ade	n=3
marm arma rmal mala alad lade	n=4

Table 5 2 Different n-grams for the word "marmalade"

One way that misspelling in queries can be accommodated in a conventional information retrieval system is by using n-grams to index both the documents and the queries When using n-grams all indexable terms are first broken up into overlapping fragments of the original term, or n-grams Before submitting a query to the search engine, it also is broken up into the same size n-grams The retrieval process then ranks the documents in the order of their similarity to the query where this similarity is calculated based on the number of n-grams which is shared between documents and query

Misspellings in user queries can also be accommodated by using string matching or dynamic programming techniques to evaluate how close a misspelled word is to a dictionary of correctly spelt words In fact, a lot of the techniques applied to matching music in different systems are based on sub-string matching techniques Some of the systems we have reviewed which employ this technique include [McNab et al , 97] and [Ghias et al , 95] These systems use approximate string matching to rank

songs based on their similarity" Approximate rather than exact string matching can be used to overcome limitations where people generate incorrect queries, although [McNab et al , 97] also allow exact string matching of melody fragments for the more musically gifted user

CEOLAIRE can use approximate or exact string matching to rank documents By default, the similarity matching is exact, which is a consequence of using *tf*idf* melody fragment weighting, but by employing the use of smaller overlapping n-grams, the system takes on the role of an approximate string matching system

When a user creates a music query, such as a query to a music information retrieval system like CEOLAIRE, errors can be introduced These errors include forgetting a note in the query, inserting an incorrect note, or repeating a note These types of errors are referred to as *insertion*, *deletion* and *duplication* and are shown in Table 5 3

Insertion	Deletion	Duplication
marmalxade	marmlade	marmalaade

Table 5 3 Examples of *insertion*, *deletion* and *duplication* errors of the word "marmalade"

Whether or not a music information retrieval system should try to accommodate and allow for these errors is debatable, but it is something that a user-friendly system should strive to achieve CEOLAIRE makes a two-fold attempt to address the issue of malformed queries First, the interface is designed to help minimise the introduction of a malformed query before it is submitted to the retrieval process with a *hear, view* and *review* strategy The user *hears* the melody of the query as it is being generated He/she can also *view* the graphical representation of the query on the paint area of the query input screen and finally the user can play the query back (*review*) before submitting it to the search engine By adding these extra features, the CEOLAIRE query handler puts a lot more emphasis on the query generation stage than one would normally find in an information retrieval system Second, the use of overlapping n-grams relaxes the requirement that a user has to enter a correct query melody fragment, but at the cost of increasing the size of the answer set It should be clear that by using smaller sized n-grams, we introduce a negative

effect on precision as new, probably non-relevant documents are introduced into the answer set There should be a minimum effect on recall as the answer set may grow to incorporate an approximately matched melody fragment, but these music files would not find their way into the already highly ranked documents Basically the high-ranking documents will continue to score highly and documents introduced as a result of smaller n-grams will rank lower and the system's ability to discriminate documents deteriorates

Analysing the corpus of music files in the collection which we have used for our music information retrieval experiments (which we discuss in detail in Chapters 6 and 7), there are a number of factors which we can observe The first thing we discover is that within the collection most notes lie between the third and fifth octaves This can can be seen in Figure 5 7 which shows the count of note occurrences where notes have been given an integer label between 1 and 72 This validates the fact that CEOLAIRE limits note detection to 72 notes for *this* dataset

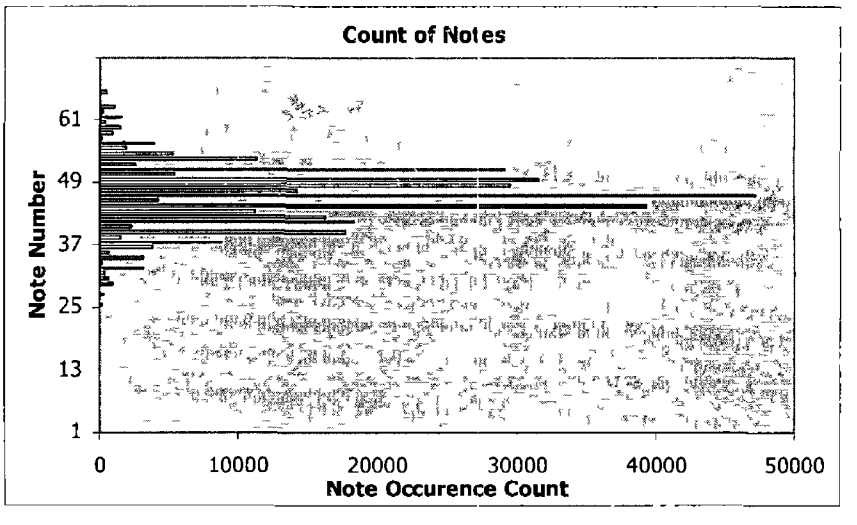


Figure 5 7 Note occurrence on the equal tempering scale

Another factor worth noting is the distribution of notes within an octave The relevance of this is to see if any note has a particular higher occurrence suggesting that perhaps weighting could be set to be slightly more biased around that note As it turns out, no note has a substantial higher share of the note count A graph of the note distribution is shown in Figure 5 8 It is worth noting that the notes represented by black keys on the piano all have a lower occurrence than those represented by

white keys

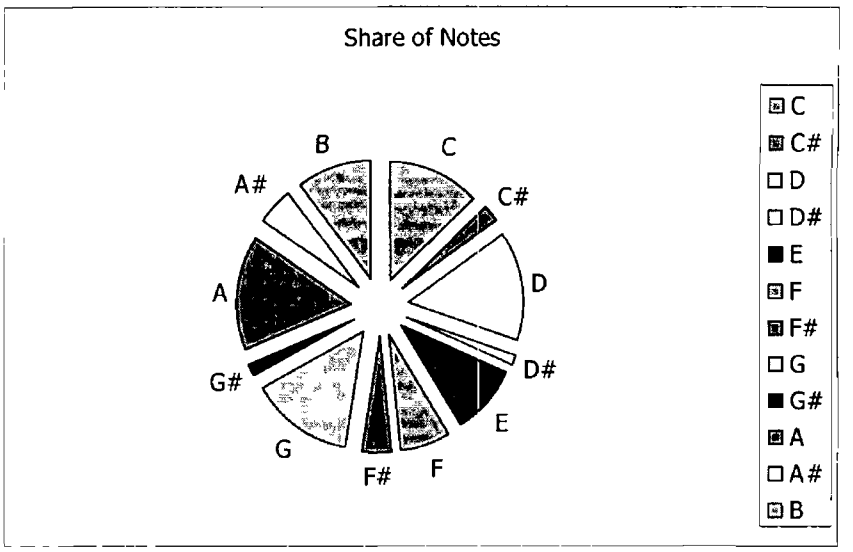


Figure 5 8 Note distribution

We can make a third observation about the dataset regarding the different interval values that occur between adjacent notes in our music database Figure 5 9 shows that most note changes occur within an interval of 5 Most note changes with an interval value greater than 12 occur with a frequency of less than 1% of all note changes Most note changes occur within 1 octave When indexing by Parsons, there *is* a loss of resolution by reducing the number of symbols available to our alphabet from 72 down to 3, but seeing as we can observe that most note changes have an interval jump less than 15 the real loss of resolution is closer to 15 down to 3 as opposed to 72 down to 3 Terms in Parsons notation draw from a finite alphabet of 3 symbols, but the alphabet could be expanded to a 5-symbol extended Parsons notation or the use of the interval could be employed Remember, when employing Parsons we are indexing based on the contour change from one note to the next If we were to extend the alphabet, it would make it more difficult for non-musically trained users to generate queries, and with an increase in the number of symbols representing music would come a corresponding increase in the number of errors which could be introduced at query time so retrieval performance could decrease For the more musically inclined users of the system we allow the system to search by Interval

In our dataset, the music file with the shortest duration is $5\frac{1}{2}$ seconds while the

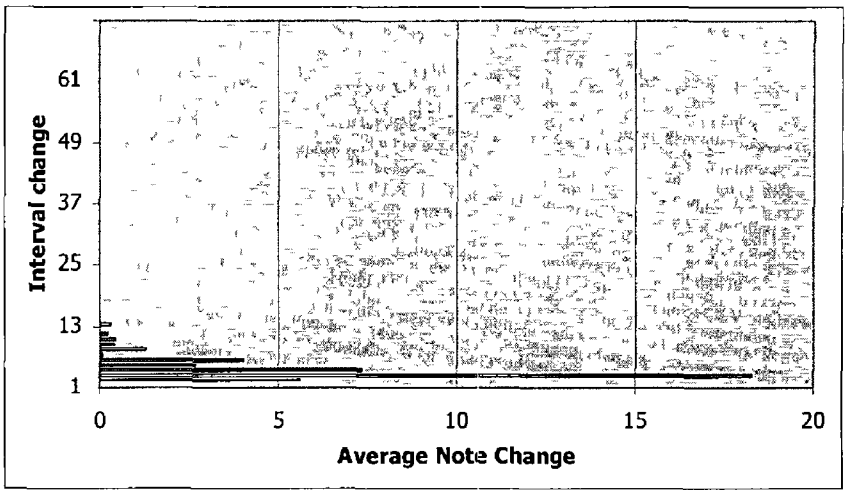


Figure 5 9 Share of note changes (Interval)

longest music file is 84 seconds The average length of a music file is 23 seconds Observing some of the files which are short we discovered that the dataset consists of both songs (in the sense that they are drawn out) and short melody snippets (in that they only lasted a few seconds) This explains the small average length of a music file It is worth noting that these observations apply only to this dataset

Another important aspect worth noting is the spatial implication of notes Terms are regarded as melody fragments, not semantically attributable tokens Within a melody fragment, the spatial locations of notes are important Using n-grams of notes preserves the spatial location of notes but, as is the case with text retrieval, the spatial location of melody fragments is lost

Table 5 4 shows some observations undertaken on the dataset, namely the number of unique n-grams One observation is that up to n-grams of size 8, the dataset contains every combination of n-grams possible, after that there is a decreasing proportion of n-gram combinations for every n-gram size The count of n-grams from 2 to 20 falls linearly, which is to be expected The ability of the system to discriminate between documents based on n-grams can also be observed from Table 5 4 by looking at the difference between unique and average unique values For example, with n-grams of size 2 and 3, the average unique count of terms per document is similar to the unique count of terms, indicating that nearly all combinations of that term appear in most documents This improves considerably up to n=20

All Parsons combinations of terms are present up to and including n-gram terms

of size 8. After that, not all the possible terms are present in the corpus.

The count of the number of terms decreases as the value for n increases. This is to be expected. Observing values for n-grams of size 3, less than 0.005% of terms are unique. Observing the values for n=20, we can see that 95% of the terms are unique. This means that there is a negative effect on precision as the answer will be a lot bigger for smaller values of n than for larger values of n. We will study this effect in more detail in Chapter 7. We also observe that the average number of unique n-grams per file rises and then falls. This is a consequence of the length of the files as shorter melody snippets do not have the length to be accommodated in the larger n-gram sizes. These figures are based on 7,299,513 terms across 9,354 documents.

n	Count	Unique	Possible Unique	Average Unique
2	483,121	9	9	8
3	473,986	27	27	17
4	464,851	81	81	26
5	455,716	243	243	32
6	446,581	729	729	36
7	437,446	2,187	2,187	38
8	428,311	6,561	6,561	40
9	419,177	19,423	19,683	41
10	410,044	52,393	59,049	45
11	400,914	110,978	177,145	51
12	391,788	178,938	531,441	58
13	382,665	237,159	1,594,323	64
14	373,545	277,802	4,782,969	67
15	364,435	301,614	14,348,907	69
16	355,332	312,756	43,046,721	70
17	346,251	315,608	129,140,163	69
18	337,192	313,600	387,420,489	68
19	328,158	308,926	1,162,261,467	67
20	319,156	302,851	3,486,784,401	66

Table 5.4: Observations of n-grams within the dataset using Parsons

Using Interval as the basis for an index, a different picture emerges. The same observations for Interval are shown in Table 5.5. The count of n-grams from 2 to 20 falls linearly, which is also to be expected. With Interval and n-grams of size 2 we have over 692 different combinations, which is a large improvement over the parsons

n	Count	Unique	Average Unique
2	483,121	692	25
3	473,986	5001	34
4	464,851	22053	38
5	455,716	65408	39
6	446,581	139,717	40
7	437,446	226,047	40
8	428,311	295,142	40
9	419,177	333,909	40
10	410,044	349,679	39
11	400,914	353,713	39
12	391,788	352,669	38
13	382,665	349,336	38
14	373,545	344,870	37
15	364,435	339,516	36
16	355,332	333,554	36
17	346,251	327,120	35
18	337,192	320,352	34
19	328,158	313,275	33
20	319,156	305,945	32

Table 5 5 Observations of n-grams within the dataset using Interval

n-gram which has 9

The count of the number of terms decreases as the value for n increases This is to be expected Observing values for n-grams of size 3, less than 2% of terms are unique Observing the values for n=20, we can see that 96% of the terms are unique Throughout the table we can see that interval has a greater number of unique terms which can only benefit document discrimination

Comparing Parsons documents to Interval documents for retrieval, we can see that the system's ability to discriminate effectively is a lot stronger for Interval More information is hidden in the Parsons UDS than in Interval, which employs a larger alphabet

Other observations, external to this dissertation yet relevant to this discussion, are based around the work of Stephen Downie [Downie, 00] using the Cranfield model for information retrieval evaluation on the same dataset as used here with information extracted from the files in the MIDI format Downe recommends the optimal length of an n-gram query, if the presence of errors is deemed irrelevant,

to be 6 To maximise fault tolerance where errors *are* deemed relevant, the length should be 4 Basically, the smaller the n-gram, the greater the fault tolerance When it comes to the length of a query a user should remember that, when generating a query, the longer the query the better, but if the system is unconcerned with query error occurrences then Downie has found that shorter queries of length 6, 8 and 10 have very good retrieval performance

Interestingly, Downie found that the location of the query within the melody of the song has no bearing on performance so the entire song should be indexed rather than incipits (the opening notes of a song) Downie's observations are based around variable length n-grams and different classifications of n-grams

Based on these observations, CEOLAIRE uses a minimum n-gram size of 4, a maximum n-gram size of 20 and indexes entire songs The upper limit can be easily be extended if needed

5 2 1 4 A Quick Introduction to Music Notation

We present a very brief introduction to music manuscript reading This is only intended to help the reader understand a worked example later in the chapter It is a simplified introduction and by no means a complete review of music manuscript reading, nor is it intended to be

Figure 5 10 shows a music staff and notes are drawn on the music staff The staff is denoted by a treble clef on the left Other clefs exist, but for the moment we will only concern ourselves with the treble clef The treble clef defines (usually) the notes above middle C, which are usually played with the right hand (on a piano) unless otherwise marked

If a note appears on a line in the staff, then it can be one of the following, E, G, B, D or F If the note appears on a space, it can be either F, A, C or E Notes can also be drawn on ledger lines, which are extra lines added above or below the staff to indicate other higher or lower notes The sharps are denoted by the occurrence of a hash (#) symbol either on the line or between the lines The same applies to flats which are denoted using the flat (♭) symbol The exact positions of the notes and the sharp symbol are shown in Figure 5 10, where we can see that F# and C#

have been defined on the music 'staff This is called *key signature*

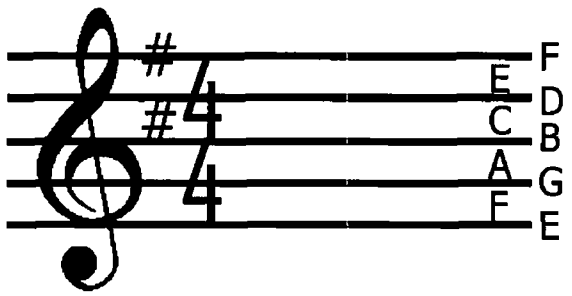


Figure 5 10 Music staff notation

The line of music is broken up into bars, represented by a vertical line through the horizontal ones Each bar has a number of beats (notes) in it The rhythm, or *time signature* is derived from the numbers on the left hand side of the staff beside the clef In Figure 5 10 the time signature is 4/4 This means that each bar contains 4 crotchet beats The occurrence of a note can be denoted by certain types of notes, for example a crotchet, a quaver or a semi-quaver Others exist, but we will limit our discussion to these and a minim, which is equal to two crotchet beats The relationship between them is that one crotchet beat can be equal to 2 quavers or 4 semi-quavers This is shown in Figure 5 11

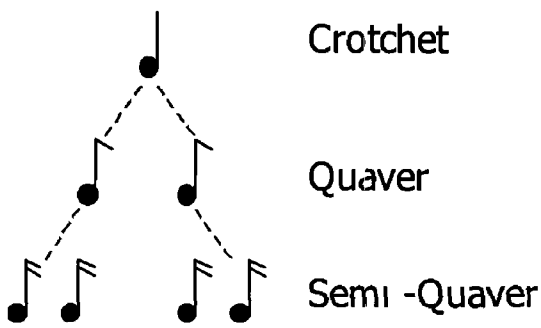


Figure 5 11 Relationship between a quaver, crotchet and a semi-quaver

The line of music is made up of bars with a number of beats in each bar The tempo of the song specifies the beats per minute (bpm) For a song with a tempo of 80bpm, then one beat should last about 3/4 of a second If the note is above the middle line, then the stem points downwards from the head, otherwise it points upwards, but there are exceptions to this rule

5 2 1 5 Extracting the Note Information in CEOLAIRE

CEOLAIRE uses a different internal representation for music than traditional music staff notation It is a lot more simplistic and less expressive The notation is not meant for trained musicians who have an intimate knowledge of music, it is a representation which helps non-musically trained users undertake retrieval Note occurrences are discovered and recorded as integer numbers in the range of 1 to 72 CEOLAIRE discards note duration, although it is available if needed Rhythm information could also be observed by the note change rate

We will now step through an example of how part of a song would be indexed by CEOLAIRE Figure 5 12 shows the first eight bars from Eric Clapton's 1991 hit song "Tears in Heaven"

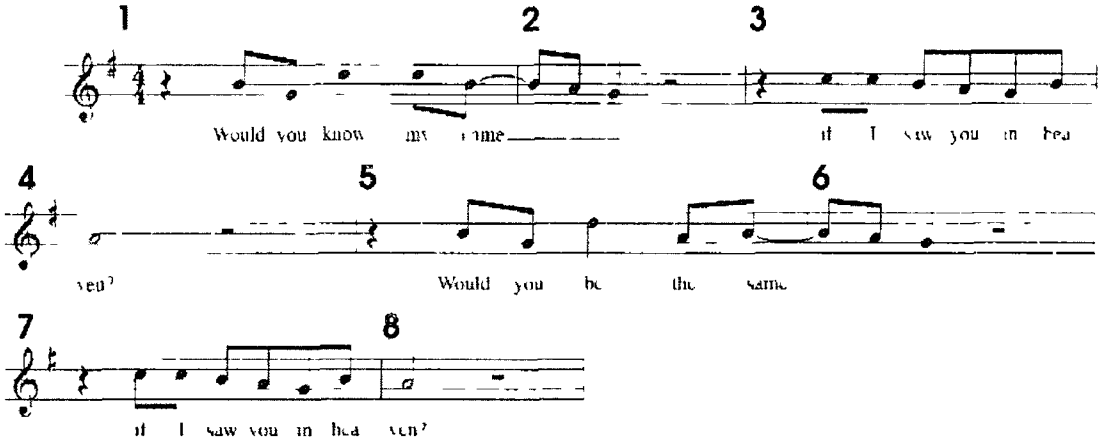


Figure 5 12 Music manuscript of "Tears in Heaven"
(c) Copyright 1991 E C Music Ltd & Blue Sky Rider Songs Ltd
Taken from "90's Hits" ISBN 0-7119-6606-0

The first thing that we observe is that the time signature defines a 4/4 rhythm Not shown is the tempo which has a value of 80 bpm Note also, here the key signature defines that all F's must be played as F# This means that the song is in the key of G As we saw earlier, every bar must contain 4 crotchet beats per bar We can translate the music manuscript one bar at a time as follows

1 The first bar starts with one crotchet beat rest, followed by two quavers, one at note B and then one at note G Next we can observe a crotchet of note D followed by two quavers, one at note D and the other at note B The fact that

note B is joined to another note B in the next bar by a line known as a tie, means that this note is is held over for the time defined by the second note, but not replayed

2 Bar two starts with the continuation of note B from the previous bar as a quaver, followed by a quaver of note A, then a crotchet of note G The bar ends with a two crotchet (also known as minim) rest

3 In bar three, we can observe a crotchet rest, followed by two quavers of note C Next, we have four quavers, at note B, note A, note G and again note B

4 Bar four starts with a minim of note A followed by a minim rest

5 Bar five starts with a crotchet rest followed by two quavers, note B then note G After that a crotchet of note D and two quavers, note A and note B Again note B is tied to the first note in bar six

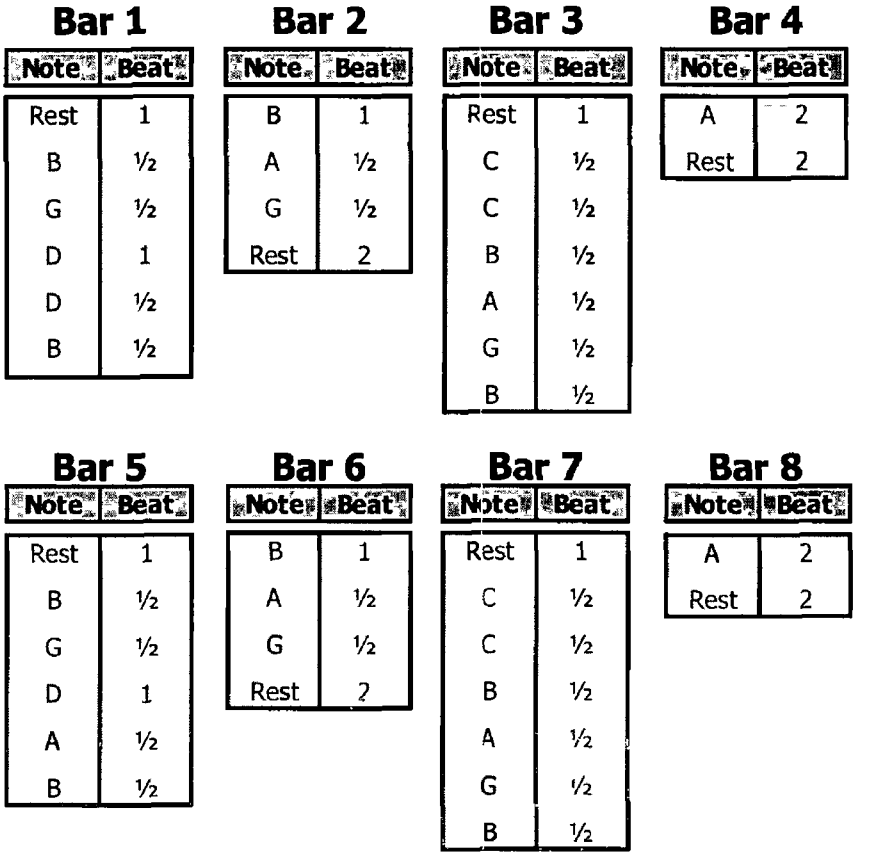
6 The first note of bar six is tied to the last note in the previous bar, followed by a quaver of note A, a crotchet of note G and a minim rest

7 Bar seven starts with a crotchet beat rest, two note C quavers and then quavers of the notes B, A, G and B

8 Finally bar eight has a minim of note A followed by a minim rest

Figure 5 13 shows the notes and their durations If this piece was rendered with a piano (for example) and the resulting rendition was recorded, it could be submitted to CEOLAIRE'S extraction engine for subsequent indexing This piece is defined using the treble clef, so we expect that the piece would be rendered within the fourth octave of CEOLAIRE'S music system CEOLAIRE'S extraction engine would then pick out the note occurrences as shown in Figure 5 14 The structure is a single long array of note identifiers

The original Equal Tempering notes, that is notes such as C, C#, D and so on are shown in Figure 5 15 Again, this is shown as a single long string of note occurrences



Bar 1

Note	Beat
Rest	1
B	½
G	½
D	1
D	½
B	½

Bar 2

Note	Beat
B	1
A	½
G	½
Rest	2

Bar 3

Note	Beat
Rest	1
C	½
C	½
B	½
A	½
G	½
B	½

Bar 4

Note	Beat
A	2
Rest	2

Bar 5

Note	Beat
Rest	1
B	½
G	½
D	1
A	½
B	½

Bar 6

Note	Beat
B	1
A	½
G	½
Rest	2

Bar 7

Note	Beat
Rest	1
C	½
C	½
B	½
A	½
G	½
B	½

Bar 8

Note	Beat
A	2
Rest	2

Figure 5 13 Translation of notes by bar

B	G	D	D	B	A	G	C	C	B	A	G	B	A	B	G	D	A	B	A	G	C	C	B	A	G	B	A

Figure 5 14 Note structure

59	55	50	50	59	57	55	48	48	59	57	55	59	57	59	55	50	57	59	57	55	48	48	59	57	55	59	57

Figure 5 15 Equal tempering notes of the notes structure

The notes are then subjected to a Parsons and Interval evaluation The Parsons evaluation examines how a note changes from one note to the next resulting in a overall melody contour sequence The length of this sequence is 26 characters and is shown in Figure 5 16 The Interval evaluation is similar, quantifying the difference between notes as both positive and negative numbers The Interval structure is shown in Figure 5 17

*	D	D	S	U	D	D	D	S	U	D	D	U	U	U	D	D	U	U	D	D	D	S	U	D	D	U	D

Figure 5 16 Parsons structure

*	-4	-5	0	9	-2	-2	-7	0	11	-2	-2	4	2	2	4	-5	7	2	-2	-2	-7	0	11	-2	-2	4	-2

Figure 5 17 Interval structure

This melody sequence is then processed into n-grams, that is an overlapping window is shifted across the string, generating sub-strings The size of the window, or the value for n varies from four to twenty The result of this process is an indexable document for each n-gram size These documents are then submitted to their corresponding search engine indexing queues for subsequent indexing An example of generating Parsons index files for n=5 and n=20 is shown in Figure 5 18 The same procedure is applied when generating Interval index files

To summarise, the CEOLAIRE system generates indexable documents from raw music monophonic files by processing each file, identifying and extracting the melody, converting this melody into Parsons notation and segmenting the overall string of Parsons notation into smaller overlapping variable length sub-strings or n-grams

5 2 2 Document Descriptors

There are three different document descriptor stores in CEOLAIRE The first is a database which stores meta-data relating to each individual music file The second is a cached copy of the music file as downloaded, in case the original link is removed Finally images of the music structure are generated automatically and stored A link to the image store and local copy of the file is kept in the meta-data database Figure 5 19 shows the three document descriptor stores

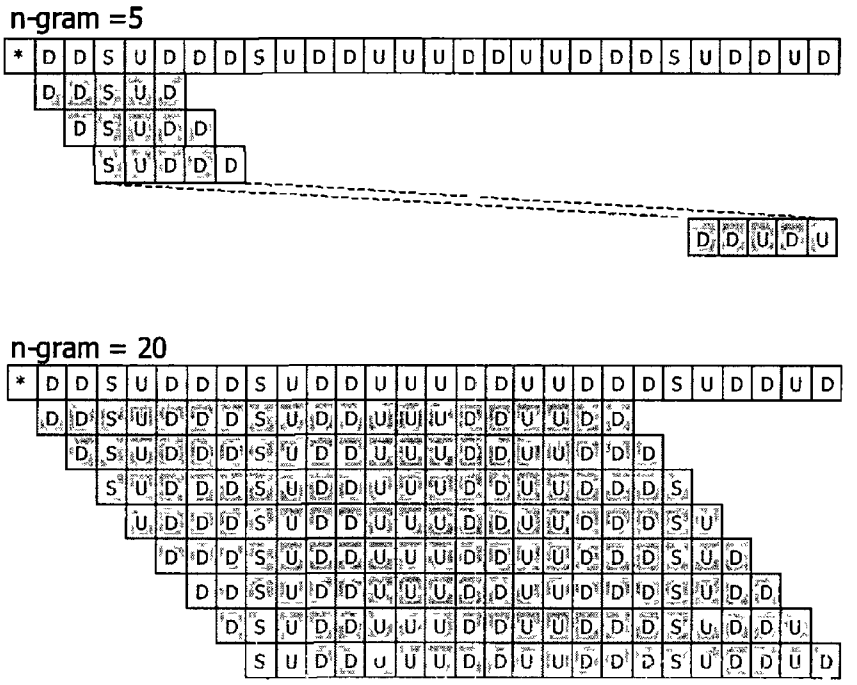


Figure 5 18 N-grammg the Parsons notation

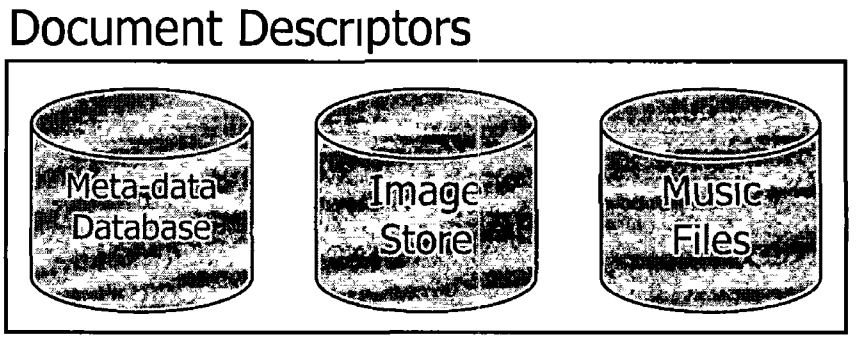


Figure 5 19 Document descriptors

5 2 2 1 Music Meta Data

The meta-data database is a Postgresql database The following meta-data attributes are retrieved and stored

- Music File Identifier (Integer Number)
- File Name
- File Format
- Sampling Rate
- Size in Bytes
- Time (Length of the file in seconds)
- Original Source Location (URL)
- Local Copy Location
- Other Information

The music file identifier is an integer which increments for every new song that we come across The name is taken from the filename The format is discovered using the Unix "file" command which can detect the format of the file Any files which are not supported are discarded Currently the system supports all the different sampling rates and bit size of the WAV format and MP3 with the use of the xaudio MP3 library [xaudio, 02] The sampling rate, file size and duration are discovered as a result of automatic meta-data extraction For MP3, the xaudio library provides an interface to achieve this Original Source Location, Local Copy Location and the Other Information field are stored in a text file along with the music file The meta-data is used at query time after the ranked list of results have been generated The query handler queries the database for each document identifier in the result set, to produce the final result set

5 2 2 2 Music Images

Music images are created for each notes file These images show how the melody contour changes over time, but are drawn as the original Equal Tempering notes Images are created on a white background with the exact notes represented as black dots Exact notes were used rather than the contour to give the user the exact representation of the song and images are stored in the png format Images are generated using the JIMI package which now comes as standard with the JDK 1 3 An example music image is shown in Figure 5 20 It is taken from the NZDL The song in question is identified as 2354

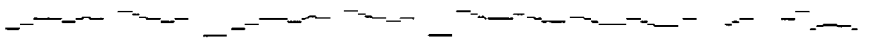


Figure 5 20 Music structure image

5.3 Search Engine

CEOLAIRE'S search engine supports both exact and approximate string matching for both contour and interval searches It utilises a number of different indices to achieve this Indices exist for both **Parsons** and **Interval** The user chooses at query time if the search is to be approximate or exact and against which indices it should be executed

The search engine is written in GNU C++ making use of the Standard Template Library (STL) wherever possible The STL is a library of advanced templates and functions whose purpose is to provide tried and tested solutions to common algorithms and data structure problems Examples include *vectors, lists, queues* and *stacks* The core of CEOLAIRE'S architecture consists of the **Indexer** which indexes documents generated as a result of the melody extraction process and the **Query Manager** which handles retrieval and other support functions The components of the search engine are shown in Figure 5 21

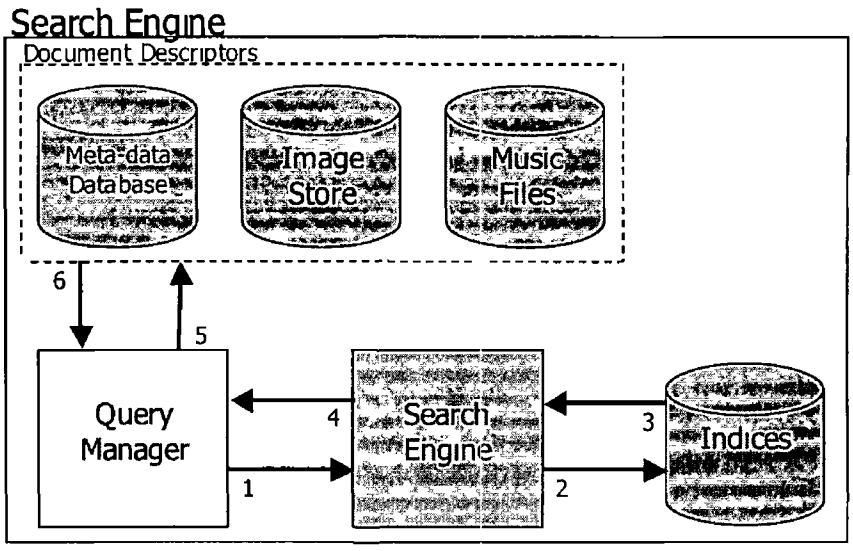


Figure 5 21 Search engine components

5 3 1 Indexing

We have seen how CEOLAIRE can transform music data into strings, changing music from its raw format into a representation suitable for indexing through the melody extraction process To CEOLAIRE, an indexable document is a text file consisting of overlapping n-grams A term is a single instance of one of these n-grams, i e "uddud" contained within a document

CEOLAIRE'S index is rebuilt when new music files are added CEOLAIRE searches the mdex-queue directory for new documents Each document consists of variable length n-grams, but there also exists a corresponding text file containing associated meta-data The meta-data is added to the database and the file descriptor associated with that file is kept for the indexer

The goal of the indexing stage is to build a Doc-Term weighted matrix Each row in the Doc-Term matrix represents a document and each column, a term A weighted Doc-Term matrix is one which has a cost weighting associated with every entry in the matrix Document vectors are created for every music document A document vector is a matrix with each row representing a term and the value of the row entry is the frequency occurrence count of a term within the document The data structures needed to build a *tf*idf* weighted index are

- Term List

- Document List
- Document Vector
- Index Vector

The Term List is a list of alphabetically sorted unique terms with a corresponding term identifier associated with each term It allows CEOLAIRE to reference term identifiers during both indexing and retrieval The Document Identifier is a similar structure assigning a unique integer identifier to each new document as it is processed It also maps the document identifier to the physical location of the document The Document Vector maps a term's frequency of occurrence within a given document A Document Vector is created for every document, and is used in the generation of the Index Vector, which maps a given document and term identifier to that terms occurrence within a document This vector is generated across the entire corpus The data structures are described in Figure 5 22

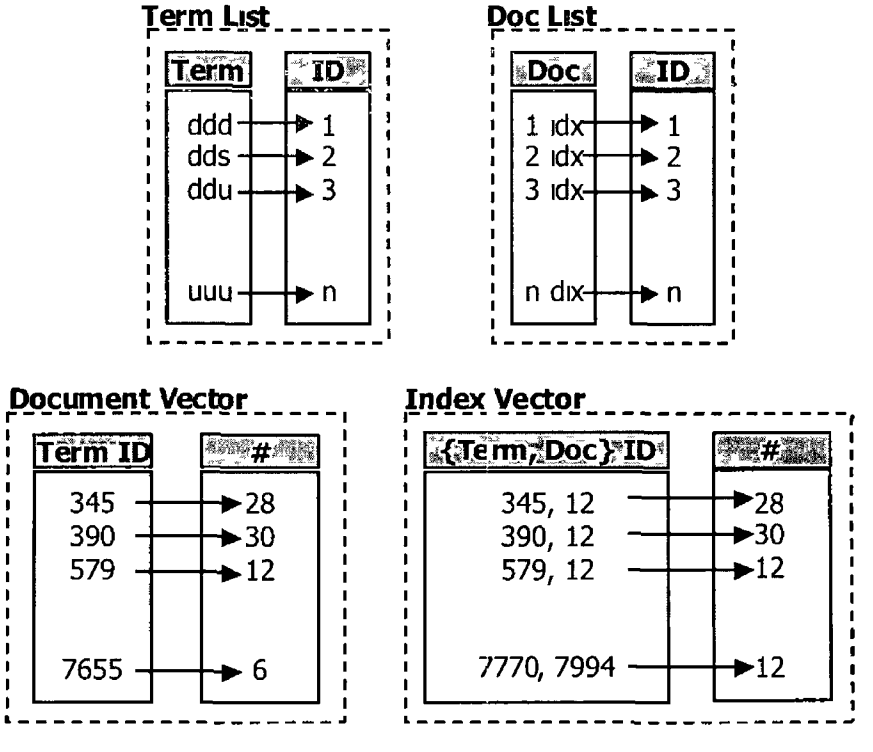


Figure 5 22 Data structures used for indexing

The Indexer is given a directory to traverse Files are processed incrementaly until all documents have been visited A processed document of music fragments as

n-grams is opened and terms are read consecutively The occurrence of a term in the current document incurs an increment of 1 which corresponds to incrementing tf_{ij} for that term The term is checked against the TermList using a binary search, checking if it has been assigned an identifier If no identifier has been assigned, then the term is inserted into the alphabetically sorted list and assigned a identifier, otherwise the identifier is retrieved and the term's identifier is added the Document Vector If this identifier exists within the document vector, its occurrence count is incremented by 1, otherwise it is added to the vector and the occurrence count is set to 1 When the current document has been processed, the document vector is used to generate the Index Vector CEOLAIRE then writes the index to disk so that the index is persistent and does not have to be generated every time the search engine is initialised The process of building the index is shown as Algorithm 3

5 3 2 Retrieval

Retrieval is achieved using a *query manager* whose role is threefold First, it reads in and initialises the retrieval engine using the Index Vector from Section 5 3 1 Second, it accepts queries from the query generator and calculates *tf*idf* weighting scores comparing the query to documents in the corpus to generate a ranked list of high scoring documents The query is a music fragment and is matched against an index of music fragments Third, for every document in the answer set, it retrieves corresponding meta-data from the database and includes the appropriate music structure image

While the query manager is running, its goal is to take queries from a user and resolve them The query manager takes a single string of Parsons notation or Interval notation and passes it to a query handler which segments it into n-grams before resolving the query If the query is to be resolved using exact matching, the string is not segmented, otherwise it is segmented into overlapping n-grams for approximate matching

One reason for making the query manager (rather than the interface) responsible for query segmentation is that it allows the interface to be independent of the retrieval process It is a lot easier to plugin new methods of query generation and

Algorithm 3 Generating the index

```
-- Variable Declaration
termID                          int
docID                           int
term                            String (variable length)
termID = 0
nexttermID = 0
docID = 0
while (more documents exist) do
 Open next document
 while (another term exists) do
   term = readNextTerm()
   if  term does not exists in Term_List{term}{?}  then
      Add term to Term_List{term}{nextTermID}
      termID = nextTermID
      nextTermID = nextTermID + 1
   else
      termID = Term_List{term}{?}
   endif
   if  termID exists in DocVector{termID}{count}  then
     DocVector{termID}{count} += 1
   else
     DocVector{termID}{count} = 1
   endif
 endwhile

 for each (DocVector{termID}{count})
   IndexVector{docID}{termID}{count} = DocVector{termID}{count}
 endforeach
 docID = docID + 1
endwhile
```

interaction into the system by treating the components as independent layers within the retrieval framework The applet submits the query string using a HTTP request calling and passing the string as a parameter to a PHP script It is this PHP script which acts as the role of query manager The query is then sent to the query handler over a TCP/IP connection which takes the query and attempts to resolve it This is outlined in Figure 5 23 Music files identified as being similar to the user's query are returned as a ranked list The output is ranked with highly scored documents first, with high scores representing the importance of the similarity between the query and the document set

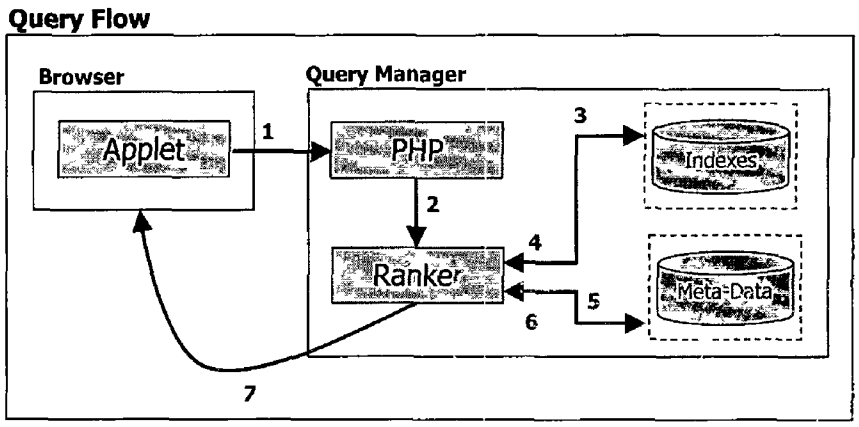


Figure 5 23 Flow of query from applet to answer set

The retrieval process has been augmented beyond that of a typical information retrieval system by allowing the user to automatically view a representation of the content of each song i e the music structure Each entry in the answer set includes a png image showing the actual note structure pertaining to a song, along with the corresponding meta-data A screenshot of the answer set generated as a result of the query "uudduu" is shown in Figure 5 24

When the search engine is initialised, it uses the Index Vector to generate a weighted Doc-Term matrix A weighted Doc-Term matrix is a two-dimensional matrix with each column representing a term and each row a document The first step of this process is to generate a Doc-Term matrix which holds the $tf_{i,j}$ values for every term and every document A Doc-Term matrix is shown in Figure 5 25 Document identifiers are shown row-wise, while term identifiers are shown column-wise

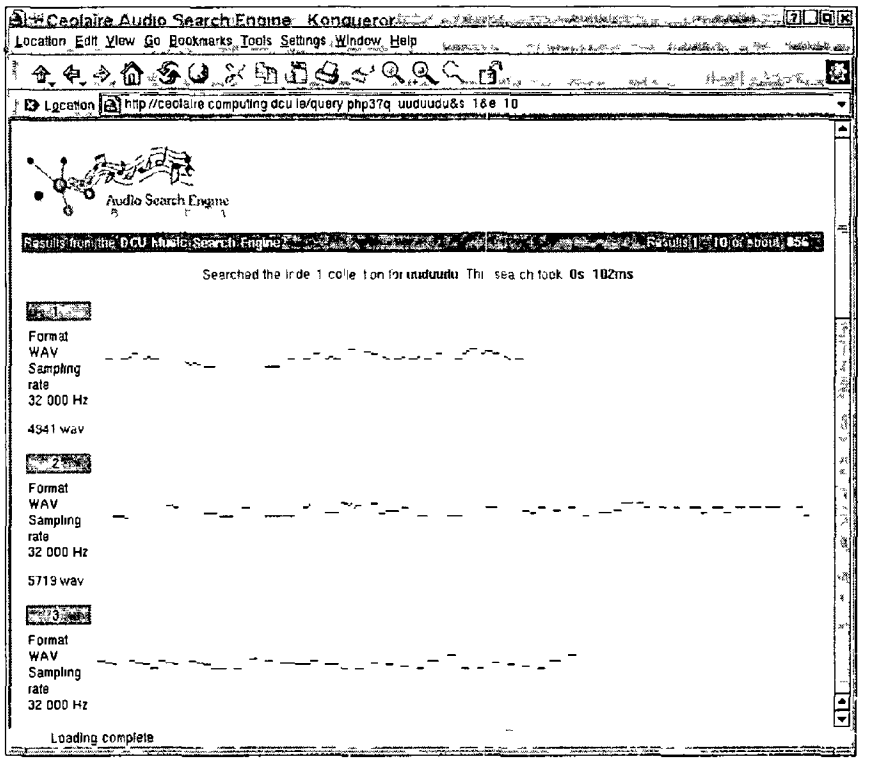


Figure 5 24 Screenshot of the ranked list of results

	T(0)	T(1)	T(2)	T(3)	T( )	T(700)	T( )	T(n-1)	T(n)
D(0)	0	8	6	5	0	3	0	3	0
D(1)	0	0	9	6	0	0	12	0	4
D(2)	12	0	0	0	1	7	6	16	0
D( )									
D(400)	5	18	0	0	4	9	0	0	6
D(401)	0	2	0	0	2	3	7	3	4
D( )									
D(n-1)	7	4	8	0	2	3	0	12	1
D(n)	8	3	7	0	0	0	1	0	3

Figure 5 25 A Doc-Term matrix

The Index Vector holds the occurrence count of a particular term within a given document for every term in every document It is processed sequentially, so for every document identifier and term identifier, each cell tf_{ij} is set to the occurrence count for that term in that document The next step of generating a weighted Doc-Term matrix is to weight each cell This is achieved using the formula $w_{ij} = 1\ 0 + log(tf_{ij})\ \log\left(\frac{N}{df_j}\right)$ Each cell is processed to produce a weighted value representing the similarity between the term and the document A weighted Doc-Term Matrix is shown in Figure 5 26

$$w_{ij} = 1\ 0 + \log(tf_{ij})\ \log\left(\frac{N}{df_j}\right)$$

	T(0)	T(1)	T(2)	T(3)	T( )	T(700)	T( )	T(n-1)	T(n)
D(0)	0	0 5	0 3	0 5	0	0 2	0 3	0 4	0
D(1)	0	0	0 4	0 4	0	0	0 7	0	0 7
D(2)	0 6	0	0	0	0 2	0 3		0 9	0
D( )									
D(400)	0 3	0 9	0	0	0 8	0 3	0	0	0 1
D(401)	0	0 3	0	0	0 4	0 5	0 4	3	0 4
D( )									
D(n-1)	0 2	0 5	0 3	0	0 7	0 9	0	0 1	0 2
D(n)	0 9	0 2	0 4	0	0	0	0 1	0	0 4

Figure 5 26 A weighted Doc-Term matrix

CEOLAIRE uses compressed storage matrices as implementing a two-dimensional matrix demands excessive storage requirements

5 3 3 Management

Music files are downloaded from the web or inserted into the processing queue automatically A cron job checks for the presence of files in the queue every night If new files are present, the index is rebuilt to include them The music files are subjected to the document pre-processing and indexing components, resulting in a new index for the search engine

5.4 User Interface

CEOLAIRE has a web interface which utilises a Java applet to aid the user when generating a music query The interface is shown in Figure 5 27 Here we can observe that middle C is marked with a small red line This can be seen on the left hand side of the screen Only four octaves are presented to the user for query formulation CEOLAIRE allows for the indexing of up to 72 notes, but we have already seen how most note changes occur within one octave so the presentation of the four octaves for query formulation is adequate As users can hear the notes while generating a query, they may have a preference as to which octave it should be composed in

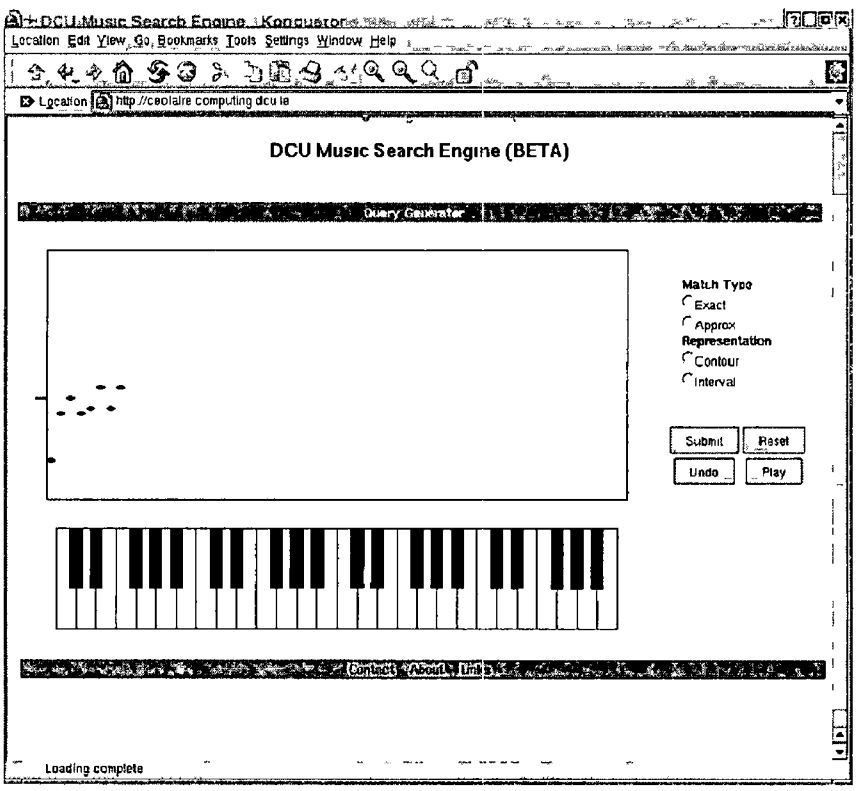


Figure 5 27 CEOLAIRE'S interface

5 4 0 1 Query Formulation

CEOLAIRE'S query generation interface helps the user when generating a query This is achieved by displaying the query graphically to the user as it is being formulated A query note is assumed to be a candidate note and is drawn in red on the painting

screen until it is committed as an actual query note The query screen is split up into three parts the **virtual keyboard**, the **paint screen** and the **command buttons** The command buttons are used to delete notes, reset the query, play the query notes and submit the query to the search engine A query can be formulated by either playing notes on the virtual keyboard or drawing the notes directly on the painting screen Two radio buttons are also presented to the user, one to choose approximate or exact matching and the other to decide if the search is to be against the Interval or Parsons index

5 4 0 2 Virtual Keyboard

When formulating a search query, the user is presented with a virtual keyboard As a user moves the mouse pointer over the keyboard the display is updated with the position of the candidate note on the paint screen This is depicted by the presence of a red circle which moves up and down as the user moves the mouse pointer over the keyboard When the user presses a key, the note is no longer assumed to be a candidate note and is committed as a query note A timer is then initiated and counts in milliseconds until the user lets go of the key When this event occurs, the end time is subtracted from the start time and the note is drawn on the paint screen Different colours and sizes are used to indicate different note durations The values are shown in Table 5 6

Duration (s)	Length (pixels)	Colour
0 > n < 1	6	Blue
1 >= n < 2	9	Green
2 >= n < 3	12	Red
n >= 3	15	Black

Table 5 6 How note durations are represented

When a key is pressed, it gives the impression that it has been pushed down The red helper ball then moves on to the next note position helping the user to guide the note to its next position Up to 28 notes can be drawn on the query screen

5 4 0 3 Paint Screen

The paint screen can also be used to manually draw a query The user achieves this by either clicking and inserting notes one at a time or by pressing a mouse button, holding it and dragging the mouse within the paint screen, drawing a contour line The paint screen can also be used to refine a query that was generated using the keyboard, by clicking on a note and dragging it either up or down

5.5 System Hardware

CEOLAIRE runs on a IBM S/390 Linux instance The S/390 in use is the 9672 model with an RB4 processor which is capable of about 63 MIPS The machine has 1 GB RAM and 800 GB of hard drive space CEOLAIRE'S virtual instance uses SuSE Linux version 7 0 The instance has 128 MB RAM and 80 GB hard drive space available to it

5.6 Summary

In this chapter we have explored the core components of CEOLAIRE, the system we built to test the effectiveness of music information retrieval and the results of which we present in the next chapters We looked at some of the important factors regarding the music collection which we have utilised, which justify the indexing approach undertaken Importantly, we saw most note changes occur within an interval of 5 We observed how a piece of manuscript music would be treated by CEOLAIRE up to the indexing stage

We have also seen how the music file pre-processing stage is used to generate png images, indexable documents by spectra generation, note-filtering and overlapping n-gram extraction Then we looked at the search engine and how the documents are indexed and retrieval is achieved Finally, we looked at the interface, introducing the different components and how they can be used to help combat the introduction of errors in a user's query From an end-users perspective, Interval matching is provided for a user with a strong knowledge of music and who has the ability to

formulate music queries with a modest amount of certainty, while Parsons matching allows the less musically inclined user undertake melody retrieval We also provide both exact and approximate matching

The main differences between CEOLAIRE and the systems we reviewed earlier is that CEOLAIRE indexes music from the WAV / MP3 file format automatically extracting the note structure Ceolaire has both approximate and exact matching capabilities Most of the previous systems work on music in the MIDI format

Next we review the methodology and implementation behind the evaluation of the core component of CEOLAIRE, the extraction engine

Chapter 6

Evaluating the Melody Extraction Engine

6.1 Introduction

In this chapter we will review the methodology and implementation behind the evaluation of one of the core components of CEOLAIRE, namely the melody extraction engine We aim to prove that the representations for music indexed by the CEOLAIRE system, which have been computed by our melody identification and extraction engine and used as part of the retrieval process, are a reasonably accurate representation of the original melody The primary goal for developing CEOLAIRE was to allow us to work with raw audio files and to research and develop retrieval techniques which stem from melody analysis at the signal level, allowing us to observe the impact of variations in our melody extraction and indexing techniques

To measure the effectiveness of CEOLAIRE and how accurately it really represents the melody being indexed in a song, we need to determine if the representation chosen for indexing (Parson and Interval) is true to the original Any evaluation of a music retrieval system should first start with an evaluation of how it generates its index and content describing structures As our index is derived from encoded digital music (as opposed to note extraction from MIDI files), we must be able to quantify to a given degree the process which automatically transcribes the encoded music into a representation suitable for indexing Thus, we need to evaluate the

performance or "correctness" of our melody identification and extraction process

CEOLAIRE underwent a number of experiments as part of this evaluation The goal was to compare CEOLAIRE'S music representation, which is derived automatically from raw audio files, against a music representation which we know to be correct This is a big problem within music information retrieval as having access to an underlying representation which we can use as a reference or ground truth, is not readily available The approach we took was to employ the use of MIDI files as the ground-truth, an approach which has also been used by other researchers when evaluating the performance of a melody extraction engine [Plumbley et al , 01] We synthesised WAV files using MIDI files which in turn are subjected to CEOLAIRE'S automatic melody extraction engine producing music representations These representations are then compared to the reference representations

Central to our evaluation process is the use of a number of different files with different representations of melodies produced as a result of CEOLAIRE'S melody extraction process

- Notes files
- Parsons files
- Duration redundant notes files

A **notes file** is generated as a result of the melody extraction process which also acts as input to the Parsons and Interval evaluation processes (this is covered in Section 5 2 1 2) Recall, the output of the extraction engine is the relative amplitude of any note that may have been found during the melody extraction process This output is stored in a file as contiguous arrays consisting of 72 floating-point numbers with the array index representing the note identifier while the corresponding element represents the amplitude The notes file is then processed to produce a new notes file consisting of overlapping n-grams of length 5 A notes file is shown in Figure 6 1 A **Parsons file** consists of a string of Parsons notation before it is subjected to the n-grammg process and a **duration redundant notes file** is a notes file which is processed to record only the contiguous onset of each new note Each representation is explained in more detail later

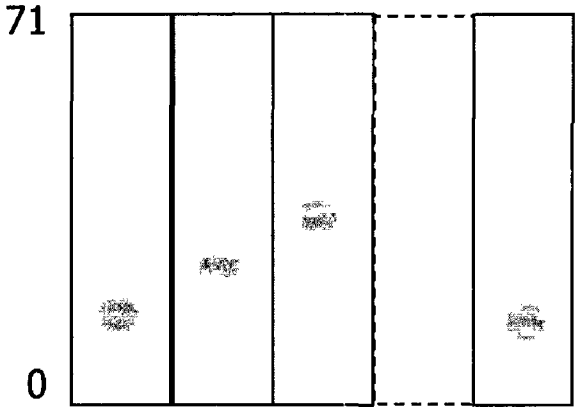


Figure 6 1 A notes file

In Section 2 6 5 we reviewed the formats and basic structure of MIDI files and how they can store music as **note on** and **note off events** These events can be readily extracted directly from a MIDI file Start and end times from MIDI files are used to generate MIDI based notes files, Parsons files and duration independent notes files Both sets of files (MIDI and WAV pairs) set their timings so that each time slice is equal to 0 05 seconds

6.2 Pre-Analysis

We have to generate suitable representations before any experiments measuring the melody extraction process can be undertaken We took our test collection from the NZDL folk-song collection This collection consists of 9,354 MIDI files Using an open source MIDI synthesiser called Timidity [Timidity, 02], we were able to synthesise 9,135 songs, but instead of playing the output on a sound card and speakers, it was captured and encoded as WAV files Timidity experienced difficulties in processing 219 of the files so they were removed from the collection This will have no notable effect on the results produced The WAV files were generated with a sampling rate of 32,000 Hz, using 16 bits per sample and in stereo which are the default parameters for Timidity

The WAV files were then subjected to CEOLAIRE'S melody extraction engine generating a notes file for each WAV file as described earlier in Section 5 2 1 3 The MIDI files were processed using a program called mftext [mftext, 02] which

Figure 6 2 Generating WAV files

generates a listing of start and end events for a MIDI file The listing produced as a result of mftext was then processed to generate notes files for each MIDI file which we then used as our ground truth against which notes files derived through CEOLAIRE processing could be compared This is shown in Figure 6 3 All the experiments undertaken to measure our melody extraction engine use these pairs of files, evaluating how similar one notes file is to the other With a collection of reference files (our ground-truth) and files produced as a result of CEOLAIRE'S melody extraction process we are able to proceed with our evaluations

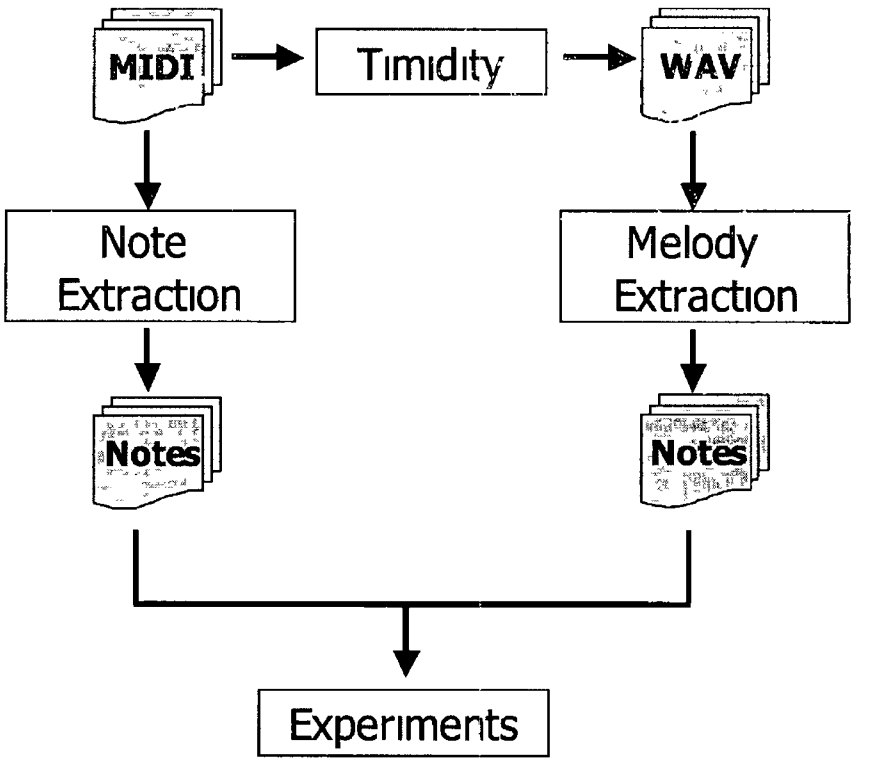


Figure 6 3 Generating MIDI-WAV pairs

6.3 Engine Evaluation

Rather than undertaking a simple difference comparison between the pairs of notes files, different processed representations of both the MIDI and WAV notes file outputs are used to generate a more informative comparison between them The motivation behind this evaluation is to quantify the extraction engine's "correctness" If it is 100% correct in identifying notes, then everywhere a MIDI note appears, we should be able to observe the presence of a CEOLAIRE derived note If this is not the case, then it does not necessarily imply that the extraction engine is incapable of doing its job Attempting a straight match is too naive, as a note could be identified as a wrong note, the extraction engine could end a note too early etc

In all, four sets of experiments were carried out to measure the melody extraction engine's level of correctness The comparison operator chosen for the experiments was the **minimal edit distance** function (covered in Section 3 3 3 7) which can evaluate similarity between two strings It achieves this by allowing the assignment of costs to the insertion, deletion and replacement of characters when converting one string into another The cost of an insertion, deletion or replacement was set to 1 for all experiments

6 3 1 Exact Note Matching

The first experiment undertaken was to evaluate the similarity between the notes file generated directly from the MIDI file and the notes file generated automatically from the WAV file, which in turn is generated from the MIDI synthesised audio of the original file This was computed based on the occurrence of precise notes and tested for approximate matches The purpose of this was to determine if, at any given time in the WAV notes file, a corresponding similar n-gram exists with its counterpart created from the MIDI file Both sets of n-grams used the same value for n, n=5 N-gram pairs were compared using the **minimal edit distance** function The distance between a pair of n-grams is divided by the length of the n-gram giving n-gram similarity The distance figures were added for every n-gram pair and this total was then divided by the total number of n-grams to give a percentage of how

close the sets of n-grams actually are This evaluation is shown in Figure 6 4

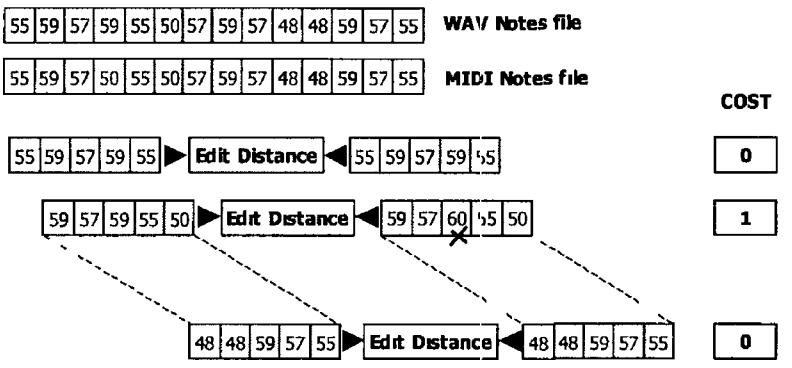


Figure 6 4 Exact note matching experiment

6 3 2 Octave Irrelevant Matching

The second experiment undertaken was to evaluate similarity after normalising all note occurrences down to one octave This was done by using the modulus operator with the note identifier and the number 12 (12 being the number of notes per octave) This brings the original alphabet of notes of size 72 down to a size of 12 by mapping each note occurrence into a single octave, or computing it modulo 12 The use of a smaller alphabet could enhance a music search engine by allowing for octave-irrelevant melody searches We have already seen that most note changes occur within one octave anyway (See Section 5 2 1 3) The note C in any octave would simply be represented as C A jump from note C in the fifth octave to note C in the sixth octave would be recorded as CC rather than C_5C_6 for such a representation of melody A query would be treated in exactly the same way at query time i e mapped to a single octave The experiment used the same costing as the first experiment The distance between a pair of n-grams is divided by the length of the n-gram giving n-gram similarity The distance figures were added for every n-gram pair and this total was then divided by the total number of n-grams to give a percentage of how close the sets of n-grams actually are The experiment is described in Figure 6 5

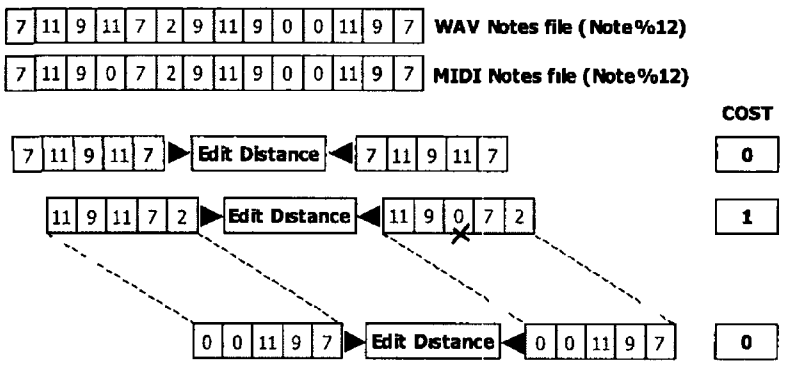


Figure 6 5 Octave irrelevant matching

6 3 3 Note Onset Matching

The third set of experiments we undertook to measure the effectiveness of the melody extraction engine was to evaluate the similarity between notes files based on exact note contour This experiment simplifies the structure of the representation of melody by ignoring the duration of notes It records only the fact that a note has changed and what that note is This experiment will show how well CEOLAIRE extracts individual notes irrespective of the note's duration This is important if the timing or duration of the original notes were to differ from that produced by the extraction engine Any user-friendly music information retrieval system should attempt to minimise the need for a user to formulate music queries with the exact length of a note as we have already seen that users have difficulty generating queries [McNab et al , 97] In our experiments, string sequences were created for both sets of notes files (WAV and MIDI derived) The minimal edit distance was used to compare the sequences and the cost was set to 1 for every edit operation The total cost of an individual MIDI-WAV notes file comparison was divided by the length of the longest sequence, giving a percentage figure indicative of the similarity between the two sequences The evaluation is shown in Figure 6 6

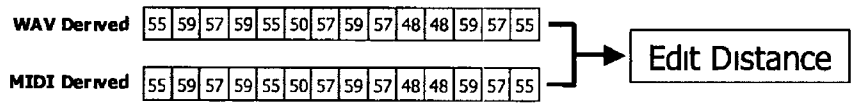


Figure 6 6 Note onsets experiment

This experiment is important as it allows us to discover if CEOLAIRE'S melody extraction finds the correct note at the correct time It can also prove that, using

Interval as an abstract music representation is a valid representation when the note onsets are correct and it logically follows that the Interval values will also be correct

6 3 4 Parsons Matching

The fourth experiment undertaken was to evaluate similarity between notes files based on note contour This experiment simplifies a notes file even further by bringing it to a representation of melody contour over time Again, actual note duration and timing information is deemed irrelevant and discarded and the representation merely indicates that a note has changed

This time the MIDI-WAV pairs of notes files are converted into Parsons notation, or up (U), down (D) or same (S), based on how the notes change [Dowling, 78] showed that the change in pitch is sufficient for reasonably accurate retrieval, thus reducing the importance of the duration of the note when it comes to melody retrieval Costs in computing the minimum edit distance similarity were the same as with the note onset experiment, with a value of 1 assigned to any of the edit operations Again, the total cost of an individual MIDI-WAV comparison was divided by the length of the longest string resulting in a percentage figure indicating the similarity between the two strings The experiment is described graphically in Figure 6 7

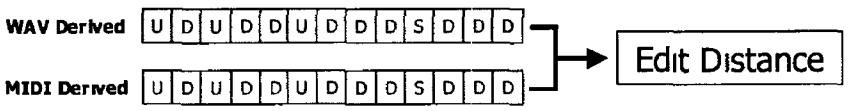


Figure 6 7 Parsons matching experiment

6 3 5 Results

The results we obtained in the first two experiments are shown in Table 6 1 From these results, we can observe that a minority of notes file pairs which emanate from the same original MIDI file are not being identified as the same The reduction of the allowable song space down to 12 notes (1 octave) in the "octave irrelevant" experiment makes a slight difference This is due to an anomaly in the extraction

engine which occurs very few times and when it occurs, it is due to the extraction engine incorrectly identifying a note in the octave above at low amplitudes As we can see from Table 6 1, 20 99% of n-gram pairs mis-match

	Exact Note	Octave Irrelevant
Number of songs	9,135	9,135
Average dissimilarity	20 99%	20 98%
Standard deviation	6 66	6 66
Number of songs with perfect match	0	0
Total number of n-grams	4,449,962	4,449,962
Total number of n-grams that match	3,559,729	3,560,050

Table 6 1 Results of exact note and octave irrelevant matching experiments

These results do not necessarily indicate that the extraction engine is doing a poor job, rather they show that some of the time the matching does not occur Analysis shows that CEOLAIRE notes end earlier than notes produced from the mftext program, but that they start at the correct time This is most likely a result of two issues First, mftext only reports onset and offset times and not the duration of notes Second, as a note is decaying its amplitude decreases, which can lead to the extraction engine not observing its presence at low amplitudes, which in turn leads to the note ending slightly early If this is the case then the results for note-onset experiment should increase significantly The matching problem is illustrated in Figure 6 8

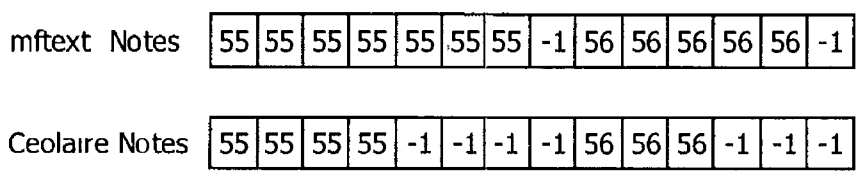


Figure 6 8 Timing problem when matching

The results from the Note Onset and Parsons experiments are shown in Table 6 2 For both the **note onset** and **Parsons** experiments we can observe a 3 27% dissimilarity The dissimilarity is due to the extraction engine identifying some notes incorrectly and the detection of repeated notes CEOLAIRE has difficulty observing repeated notes played quickly as there is no boundary between the notes Figure 6 9 shows an example of how repeated notes can cause a problem Post processing

the notes observing amplitude decline can help to detect note boundaries

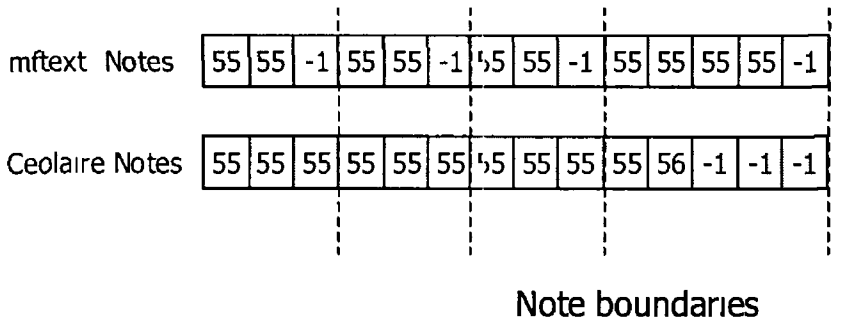


Figure 6 9 Problem with repeating notes

This suggests that the CEOLAIRE melody extraction engine works reasonably well for note onset and Parsons generation This also confirms our earlier theory regarding the duration of notes The fact that simplifying the representation even further using Parsons offers no increase in performance indicates that when the extraction engine is wrong, it is consistently wrong and no extra similarity benefits are to be gained by using an exact note based representation rather than a Parsons representation The note-onset and Parsons representation currently do not utilise timing information with respect to the length of notes when generating the index and, as such, the extraction engine is more concerned with extracting note onsets

	Note Onset	Parsons
Number of songs	9,135	9,135
Average dissimilarity	3 27%	3 27%
Standard deviation	9 1	9 1
Number of songs with perfect match	6,468	6,468

Table 6 2 Results of note onset and parsons experiments

The note boundary problem may be resolved by building another layer on top of the extraction engine which analyses the power dissipation of musical notes as they are played This would make it easier to detect repeated notes Another approach which could be undertaken would be to not index repeated notes This would entail the note changes being recorded as either up (U) or down (D) This is an approach we would rather not pursue as it incurs a further loss of resolution which would have a negative effect on precision

6.4 CEOLAIRE and MP3

The storage requirement for the dataset in the WAV format is around 28 GB for 62 hours and 21 minutes of music which is rather excessive since there is no compression of the audio files used at all If we were to build and deploy a music retrieval system which allowed streaming of the audio or allowed people to download the music files, we would have to store it in a format which has low storage requirements and acceptable sound quality MP3 files (Section 2 6 7) are adequate for this purpose and have recently become very popular, so we undertook another experiment to evaluate whether encoding music files using the MP3 format would lead to any performance degradation with respect to the melody extraction process MP3 is a lossy codec and our ears may not be susceptible to minor fluctuations which may occur during the codec's operation, but how will CEOLAIRE deal with it? These experiments can tell us if, within CEOLAIRE, the format is suitable for indexing or solely for storage

The first step undertaken as part of the experiment was to generate MP3 files from the collection of WAV files used in the previous experiment These MP3 files were then converted back into WAV files and the same set of experiments as above were carried out The process is outlined in Figure 6 10

The MP3 files were generated using the open-source MP3 Blade encoder (v0 76) Each WAV music file was converted into MP3 at the following bit rates 32 kbps, 64 kbps and 128 kbps Once they were encoded they were decoded back into WAV files using the mpg123 [mpg123, 02] MP3 decoder with sox [sox, 02]

Three sets of experiments were undertaken to measure the impact of encoding our music files in MP3 The results of these experiments are summarised in Table 6 3 From this we can see that the quality of the MP3 encoding at 128 kbps and 64 kbps is sufficient to allow its use for direct indexing of melody With MP3 files encoded at 32 kbps, the results are slightly more problematic as they lead to a decrease in melody extraction effectiveness The quality of the audio is low anyway and an aural observation confirms the presence of distortion which detracts from the extraction engine's ability to reliably detect notes Although, despite the distortion, CEOLAIRE is able to detect most of the notes with a small degree of error

The biggest impact of using MP3 as the encoding storage format for our audio

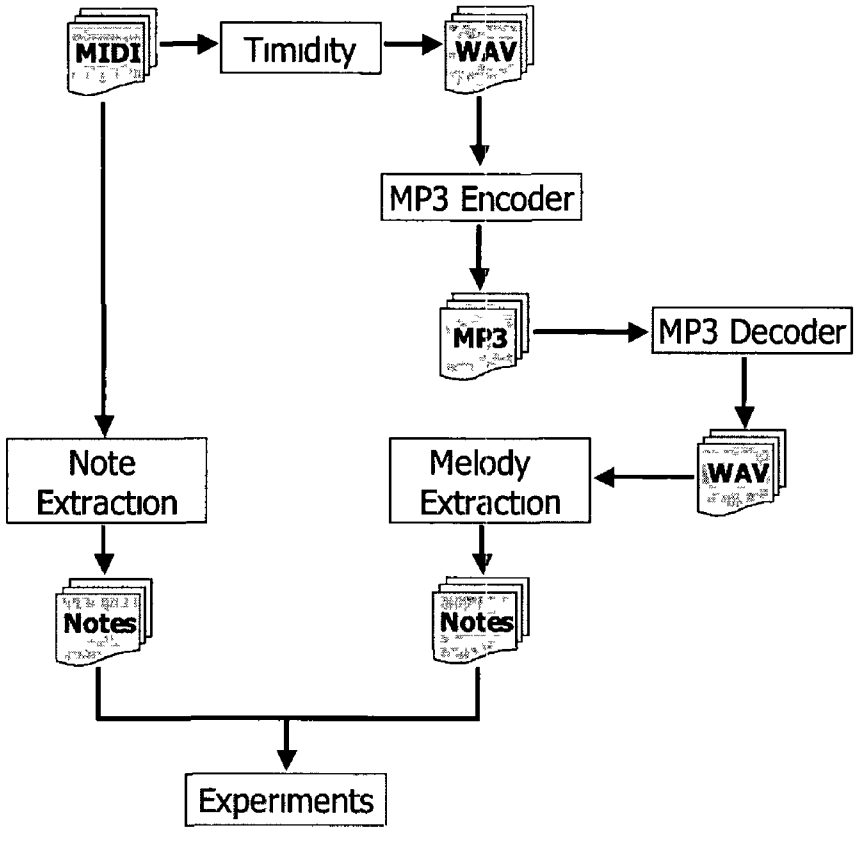


Figure 6 10 Evaluating MP3 files for use within CEOLAIRE

Results	128 Kbps	64 Kbps	32 Kbps
Exact Note	20 55%	21 45%	23 17%
Octave Irrelevant	20 54%	21 44%	23 16%
Note Onset	3 27%	3 27%	4 26%
Parsons	3 27%	3 27%	4 26%

Table 6 3 Results of MP3 derived experiments

files is in terms of storage costs Instead of requiring 28 GB to store 62 hours and 21 minutes of music, this is reduced hugely as Table 6 4 shows

Clearly the impact of using compressed MP3 is very significant and, as Table 6 3 shows, there is no loss of accuracy in our melody extraction with this storage format From this we can infer that music files compressed using MP3 at 128 kbps and 64 kbps are of sufficient quality to be used for indexing, storage and streaming, while the distortion in files compressed at 32 kbps result in a decrease in performance

Storage cost (bytes)	WAV	MP3 (128)	MP3 (64)	MP3 (32)
Total Storage	28,737,826,284	3,435,946,313	1,776,298,328	895,464,809
Average Storage	3,145,903	356,588	194,449	98,025

Table 6 4 Reduction in storage costs using the MP3 file format

6.5 Summary

This chapter presented a series of experiments which measured the effectiveness of the CEOLAIRE melody extraction engine The results show that CEOLAIRE works reasonably well on automatic extraction of melody from monophonic music sources The way in which the experiments were run may seem somewhat of an over-simplification as in our series of experiments we appear to have simplified the similarity measure we used until we obtained strong enough similarities between notes files generated from the same MIDI sources but using two different approaches This approach is justified by us when we consider that others [McNab et al , 97], [Beeferman, 97] have already shown (when working with MIDI files) that such melody contours are a sufficient basis for good quality retrieval from music Using MIDI as a format for generating PCM files for comparison purposes has also been carried out by [Plumbley et al , 01]

As a result of these experiments we have shown that our choice of using Parsons and Interval as representations for music is justified as the melody extraction engine can create an abstraction of the original music files which is suitable for indexing and reasonably true to the original music source We have also seen that the use of MP3 files encoded at 64 kbps and 128 kbps using the Blade Encoder is suitable as a format for both indexing and storage as the MP3 format manages to act as a reasonably accurate source for our melody extraction

It is clear that, when developing an extraction engine for music, an evaluation must be carried out to allow it to be refined and improved upon The experiments reported in this chapter show that the melody extraction engine in CEOLAIRE is good enough to calculate the melodic contour and even the exact note melodic contour with enough accuracy to allow subsequent retrieval Having observed that our structures are not 100% accurate, the next step is to evaluate the impact this

has on retrieval effectiveness

Chapter 7

Retrieval Performance Evaluation

7.1 Introduction

In this chapter we present the methodology and results of our evaluation of the effectiveness of CEOLAIRE as a music information retrieval system It is known that users of music information retrieval systems have difficulty generating correct music queries, a fact which has been observed by [McNab et al , 97] and therefore our evaluation process is only concerned with the system's retrieval performance, not people's ability to generate correct queries If we were to include user evaluation using real people, we would first have to undertake the evaluation which is explained in this chapter These results would then become baseline retrieval results against which we could test different interfaces A suggested interface would be something similar to that of Ceolaire's but other interface examples could be query-by-humming, whistling etc Even a simple text box which a user has to enter Parsons or Interval manually Another interface which could be developed would work on top of Ceolaire's and would have a query-by-example component where the Interface aides the user by 'auto-completing' the query for the user, eliminating the need to recall the entire melody

There are a number of different aspects of CEOLAIRE which we wish to evaluate the system's performance compared against a reference system, the effect that varying the size of the n-gram has on retrieval performance and finally, if there is a particular combination of n-grams which lead to increased retrieval performance

7.2 TREC Evaluation

The Text Retrieval Conference (TREC) [TREC, 02] is a conference which is held annually and provides researchers with a platform upon which they can evaluate and collaborate on information retrieval techniques For its participants, TREC has a number of different tracks and, for each track, sets a number of tasks which involve the generation of a ranked list of document identifiers Participants take and complete the tasks using various retrieval mechanisms and techniques, the result of which is a ranked list of document identifiers This ranked list of document identifiers is evaluated using a package called **trec_eval** [TREC_EVAL, 02] which reports average precision at various cut-off points as well as single value summary measures which are themselves derived from the precision and recall figures In Section 1 6 we saw that **Precision** is the count of relevant documents in the answer set divided by the count of documents in the answer set indicating the proportion of retrieved documents which are relevant while **Recall** is the count of relevant documents in the answer set divided by the count of relevant documents in the corpus showing the proportion of relevant documents which have been retrieved The cut-off points show average precision figures at various levels of recall, for example, a cut-off point of 5 implies that we are observing the average precision of the top 5 documents Precision and recall graphs are then generated in order to visualise retrieval performance as well as providing a method for comparison of retrieval runs

Central to evaluation undertaken using the trec_eval package is the use of a **RelDocs** listing and a **DocRank** listing The RelDocs listing is a catalogue of query identifiers and corresponding document identifiers which have been judged as relevant to the queries In a normal TREC evaluation scenario (using a TREC test collection) the RelDocs listing is generated using a pooling technique, whereby diverse retrieval systems suggest documents for human evaluation and this allows incomplete relevance judgements to be generated, which are then used as a basis for RelDocs generation A DocRank listing is produced as a result of taking each query and submitting it to an information retrieval system which generates a ranked list of highly scoring documents The trec_eval package is then employed to evaluate the two listings, generating precision and recall values at various cut-off points which are

indicative of the performance of the information retrieval system under investigation Evaluation using the trec_eval package is illustrated in Figure 7 1

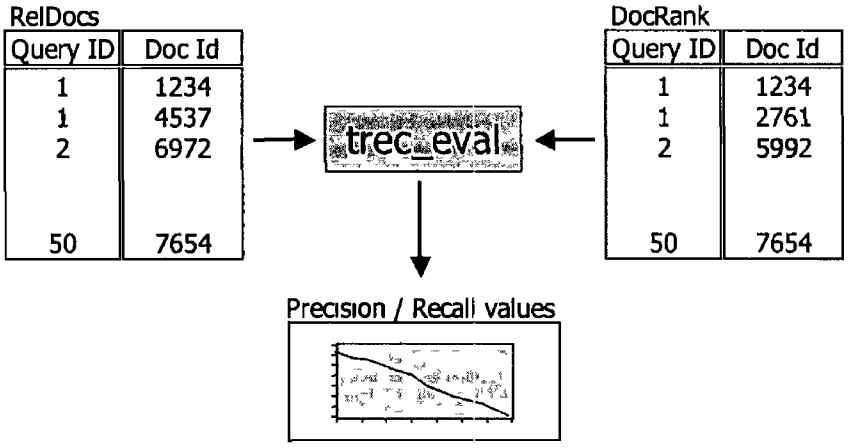


Figure 7 1 TREC eval

Within the scope of TREC, the use of trec_eval is a standard method for evaluating the performance of an information retrieval system and has been used continuously over the last number of years to evaluate the performance of information retrieval systems in various TREC tracks Some of the tracks undertaken as part of TREC include topic searching, ad hoc retrieval, filtering and emerging specialist information retrieval applications such as multi-lingual information retrieval [Smeaton, 98]

7.3 Performance Evaluation

The method we chose to evaluate the performance of our music search engine is based on the trec_eval model described earlier, primarily because it is a standard retrieval performance evaluation methodology accepted by the information retrieval community First, we generated music queries or sub-strings of music fragments in order to produce a RelDocs listing Only one RelDocs listing is created and used among all experiments 50 real music fragments were generated manually on a keyboard by a person with music knowledge before being transcribed into Parsons notation, after which the occurrence of each Parsons fragment in the dataset was confirmed by submitting the Parsons fragment to CEOLAIRE and observing

the answer set The music fragments and their Parsons equivalents are shown in Appendix A The answer set (the list of document identifiers which are relevant to each query) was used to generate the RelDocs listing The selection technique which we employ, namely using sub-strings (or terms) from within the database under investigation is a valid method for generating experimental queries [Tague-Sutcliffe, 92] and has also been used by Downie in his experiments on music n-gram lengths [Downie, 00] Downie needed a number of query terms which he could use for his experiments so he chose to take them from within the database as he knew those queries would be relevant The TREC conference supplies a list of documents which are relevant to a set of queries This approach is an easy method for generating a list of relevant documents for a set of queries without the need for human relevance judgements This led to the generation of a RelDocs listing which we could use as input to trec_eval for evaluation purposes This process is shown in Figure 7 2

Three different experiments were carried out in order to evaluate CEOLAIRE'S retrieval performance First, we **quantify the effect** on retrieval performance as a result of using an index derived from our melody extraction process as opposed to simple note extraction from MIDI files and second, we observe the performance of the search engine when retrieving documents under both **exact matching** and **approximate matching** conditions Finally, we observe whether a particular **combination of n-grams** can lead to an increase in retrieval performance The source music files are again the MIDI files taken from the folk song collection of the NZDL library and the data structures used for all experiments are strings of Parsons notation

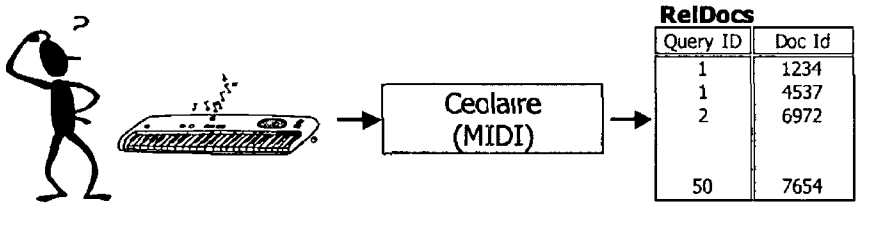


Figure 7 2 Generating the RelDocs listing

7 3 1 Baseline Performance

The first experiment uses the trec_eval package to compare the retrieval performance on two music corpora The first music corpus is generated as a result of Parsons notation extraction from the original MIDI files and the second as a result of CEOLAIRE'S melody extraction process (again the WAV files were synthesised from the MIDI files) We will refer to these corpora as **CEOLAIRE MIDI** and **CEOLAIRE WAV** respectively Both corpora were indexed using CEOLAIRE We have already seen in Section 6 3 how we can compare representations taken directly from the original MIDI files against representations generated as a result of our melody extraction process in order to observe the performance of the extraction engine Again, we used MIDI derived structures as an upper-bound, against which we compared our own automatically derived structures Although, this time we observed the performance of the search engine using the CEOLAIRE WAV corpus against the performance of the search engine using the CEOLAIRE MIDI corpus in order to identify any effects that our melody extraction process may have had on retrieval effectiveness The steps involved to complete this experiment are shown in Figure 7 3

The experiment was implemented by employing the use of the Timidity [Timidity, 02] synthesiser to generate WAV files for each MIDI file, which in turn were submitted to CEOLAIRE'S melody extraction process, generating the CEOLAIRE WAV index The CEOLAIRE MIDI index was also generated by extracting the note information directly from the MIDI files The 50 queries were submitted to both versions of CEOLAIRE in batch mode which resolves the queries and generates DocRank listings The DocRank listings were then used in conjunction with the original RelDocs listing as input to the trec_eval package generating Precision / Recall figures for comparison, which is indicative of the retrieval performance of the system

The results from this experiment are shown in Figure 7 4 as a Precision / Recall graph We observe a decreasing line representing the retrieval effectiveness of the CEOLAIRE WAV index which is derived from our melody extraction process We know from Chapter 5 that there are issues relating to the effectiveness of the extraction engine and this type of evaluation allows us to quantify how it impacts on

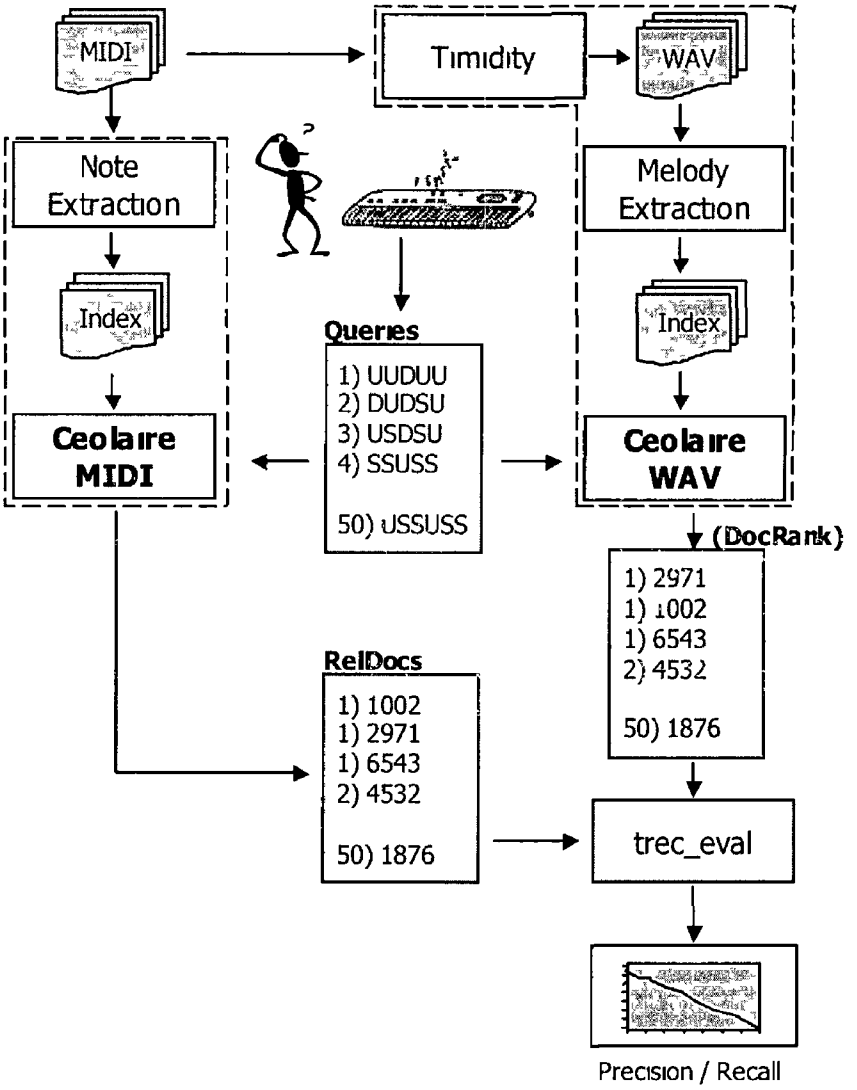


Figure 7 3 Steps undertaken to observe the performance of the CEOLAIRE WAV index against the CEOLAIRE MIDI index

retrieval effectiveness The figure demonstrates the decreasing performance of the CEOLAIRE WAV index The reason for this deterioration is related to the difficulty of observing note boundaries which is covered in Section 6 3 5 We are comparing this performance against the CEOLAIRE MIDI index If we were to draw the performance of the the CEOLAIRE MIDI index, it would be drawn as straight horizontal line with a precision of 1 as it is the reference index

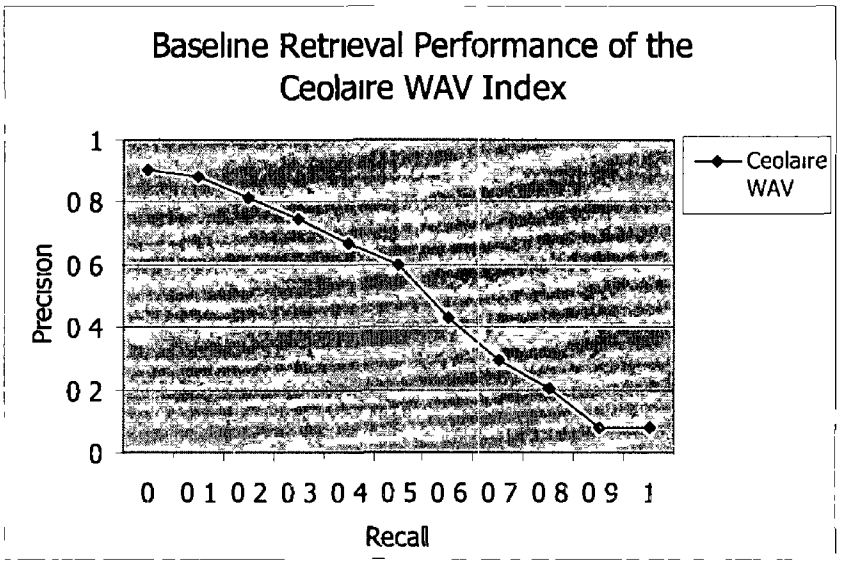


Figure 7 4 Observing the baseline performance of the CEOLAIRE WAV index

CEOLAIRE'S retrieval effectiveness is also shown in Table 7 1 with precision at 5, 10 and 20 retrieved documents or music files This implies that 64% of the top 5, 51% of the top 10 and only 33% of top 20 retrieved documents are relevant The average precision of the CEOLAIRE WAV index is 0 4941 Again, this deterioration in retrieval is due to the difficulty of observing note boundaries, but we have now qualified its impact on retrieval effectiveness

Precision at	5	10	20
Ceolaire WAV	0 6400	0 5100	0 3310

Table 7 1 Precision at 5, 10 and 20 documents when observing the baseline performance of the CEOLAIRE WAV index

7.3 2 Approximate Versus Exact Performance

In this experiment we observe the system's retrieval performance under two conditions, the first being **exact matching** and the second **approximate matching** Exact matching can be undertaken by not segmenting the query string into smaller sub-strings or n-grams before submitting it to the search engine while approximate matching is achieved by segmenting the query into smaller sub-strings before submission The length of the shortest and longest query n-grams are 8 and 12 respectively The distribution of query lengths is shown in Table 7 2 Employing the use of queries longer than the shortest query (size 8) may skew the results as not all queries will be present during the different retrieval runs We can compare the results obtained for n-grams of size 8 for all the queries If we were to use n-grams with a size larger than 8 we would be using only a subset of the entire set of queries and any results obtained would only be applicable that subset of queries and thus we would not be comparing like with like Therefore, we set an upper limit of 8 characters for the size of a query n-gram for the experiment

Query length	8	9	10	11	12
Count	3	21	14	9	3
Percentage	6%	42%	28%	18%	6%

Table 7 2 Query lengths

The experiment employed the use of n-gram sizes from 2 up to 8 and in all, 7 retrieval performance runs were undertaken We apply Boolean OR logic when resolving queries because the use of AND logic when performing approximate string matching is impractical The steps undertaken to implement this experiment are illustrated in Figure 7 5

Recall, we are using an index generated as a result of the melody extraction process on WAV files created using the Timidity MIDI synthesiser, but this time we are only concerned with the retrieval performance of the CEOLAIRE WAV index For this experiment the 50 queries were submitted multiple times (once for each n-gram size) to the search engine Each time they were submitted the size of the n-gram increased by 1 DocRank listings were generated from the output of each run and using the original RelDocs listing, trec_eval was employed to generate Precision /

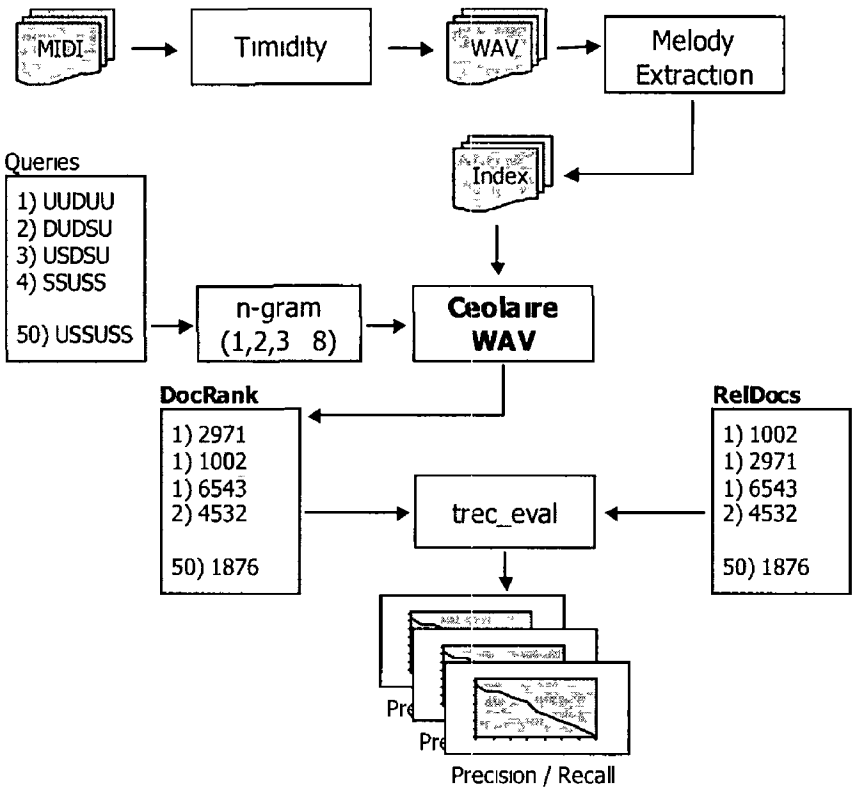


Figure 7 5 Steps undertaken to observe the retrieval performance of the CEOLAIRE WAV index using variable length n-grams

Recall values

The results from each run are plotted against each other as the Precision / Recall graph shown in Figure 7 6 Precision at 5, 10 and 20 documents for each retrieval run are listed in Table 7 3 Average precision is shown in Figure 7 7

Precision at	5	10	20
n=2	0 0040	0 0100	0 0060
n=3	0 0160	0 0200	0 0160
n=4	0 0440	0 0440	0 0400
n=5	0 1360	0 1100	0 9400
n=6	0 2080	0 1940	0 1910
n=7	0 3560	0 3280	0 2540
n=8	0 5080	0 4480	0 3060
Baseline	**0 6400**	**0 5100**	**0 3310**

Table 7 3 Precision at 5, 10 and 20 documents when observing the performance of the CEOLAIRE WAV index as an approximate matching system

From Table 7 3 and Figure 7 7, we can observe that retrieval performance deteriorates as the size of n decreases This performance deterioration is as a result of

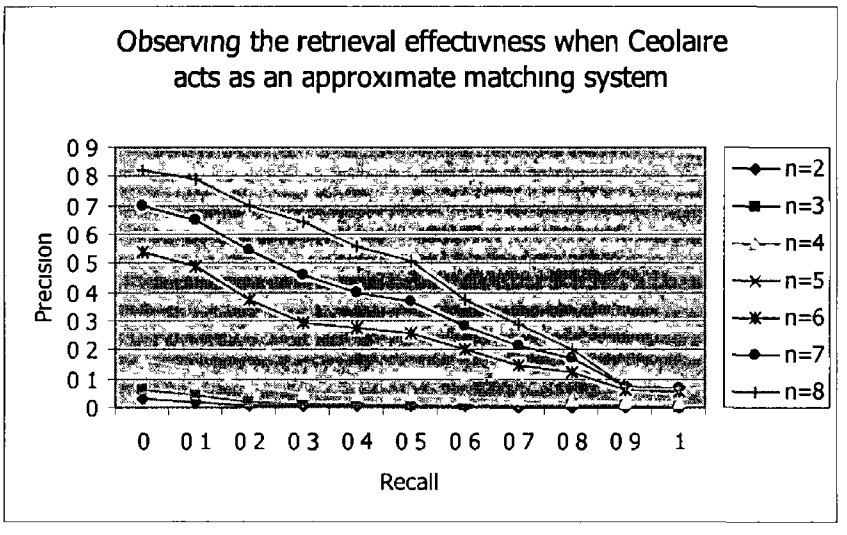


Figure 7 6 Precision and Recall of the CEOLAIRE WAV index as a result of varying the length of the n-grams

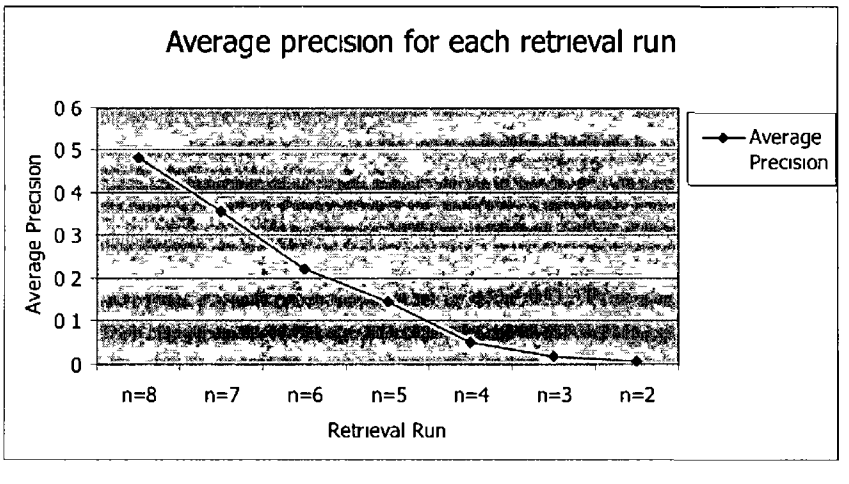


Figure 7 7 Average precision for each retrieval run

new non-relevant documents being introduced into the answer set Precision suffers as the size of the answer set grows with the introduction of new non-relevant documents Each time the size of the n-gram decreases, different sets of non-relevant documents are introduced into the answer set and this results in a decrease m performance Comparison between the baseline precision figures and the precision figures from this experiment demonstrates a consistent drop m retrieval effectiveness In particular, if we compare the baseline precision figure at 5 docs to the precision figure of n=8 at 5 docs, there is a drop in precision of 132 This shows us the cost of using CEOLAIRE as an approximate matching system The precision at 5 docs for n=8 is 127 times better than precision at 5 docs for n=2 which shows that decreasing the size of the n-gram down to 2 has a dramatic effect on retrieval effectiveness

7 3 3 Combination Performance

The third experiment relating to retrieval effectiveness is based around the idea of data fusion Data fusion is a technique whereby the outputs of various retrieval strategies are combined to yield a level of retrieval effectiveness which is better than any of the individual retrieval strategies This technique has been employed with success by researchers such as Smeaton [Smeaton, 98] and Lee [Lee, 97] m order to observe how the combination of independent retrieval strategies can increase the effectiveness of retrieval performance Lee observed that, when data fusion increased performance, the different retrieval runs discovered similar sets of relevant documents but different sets of non-relevant documents We observed the same in our previous experiment Smeaton experimented on Spanish texts taken from the TREC 4 Spanish documents experiments and found that data fusion does indeed yield improved retrieval results m the case where the retrieval strategies are truly independent of each other

There are 2 methods which may be employed when evaluating the effectiveness of data fusion The evaluation can express the same information need as a query to each system under investigation and compare the answer sets Alternatively, the evaluation can fuse the ranked output of document identifiers generated by the various retrieval systems

For this experiment we take a similar approach to data fusion and ask the question "*Is there a combination of n-grams which may lead to an increase in retrieval performance?*" We treat each combination of n-grams as a different retrieval technique and although these techniques are **not** independent of each other, we wish to observe what effect, if any, combining these techniques have on retrieval effectiveness The combinations we chose are basically n-grams of n-grams and are shown in Figure 7 8

RunID	2
1	2, 3
2	3, 4
3	4, 5
4	5, 6
5	6, 7
6	7, 8

RunID	3
7	2, 3, 4
8	3, 4, 5
9	4, 5, 6
10	5, 6, 7
11	6, 7, 8

RunID	4
12	2, 3, 4, 5
13	3, 4, 5, 6
14	4, 5, 6, 7
15	5, 6, 7, 8

RunID	5
16	2, 3, 4, 5, 6
17	3, 4, 5, 6 ,7
18	4, 5, 6, 7, 8

RunID	6
19	2, 3, 4, 5, 6, 7
20	3, 4, 5, 6, 7, 8

RunID	7
21	2, 3, 4, 5, 6, 7, 8

Figure 7 8 Combining various size of n-grams

We employ the method chosen by Lee, namely to express the query multiple times to the different retrieval strategies The steps undertaken to achieve the experiment are shown in Figure 7 9 The process is similar to the previous experiment, with the difference being how the query strings are segmented into n-grams

The queries were broken up into n-grams in the combinations shown in Figure 7 8 and in total, 21 retrieval runs were undertaken Instead of graphing the results from the various runs together into one big graph, results are grouped by increasing sizes of n-grams of n-gram retrieval runs The grouped results of these retrieval runs are shown in Figure 7 10 as 6 Precision / Recall graphs plotting each run against the others in its group

For each grouping, the combination increases retrieval performance as the size of the grouped n-grams is increased by 1, a fact which can be observed across all the graphs

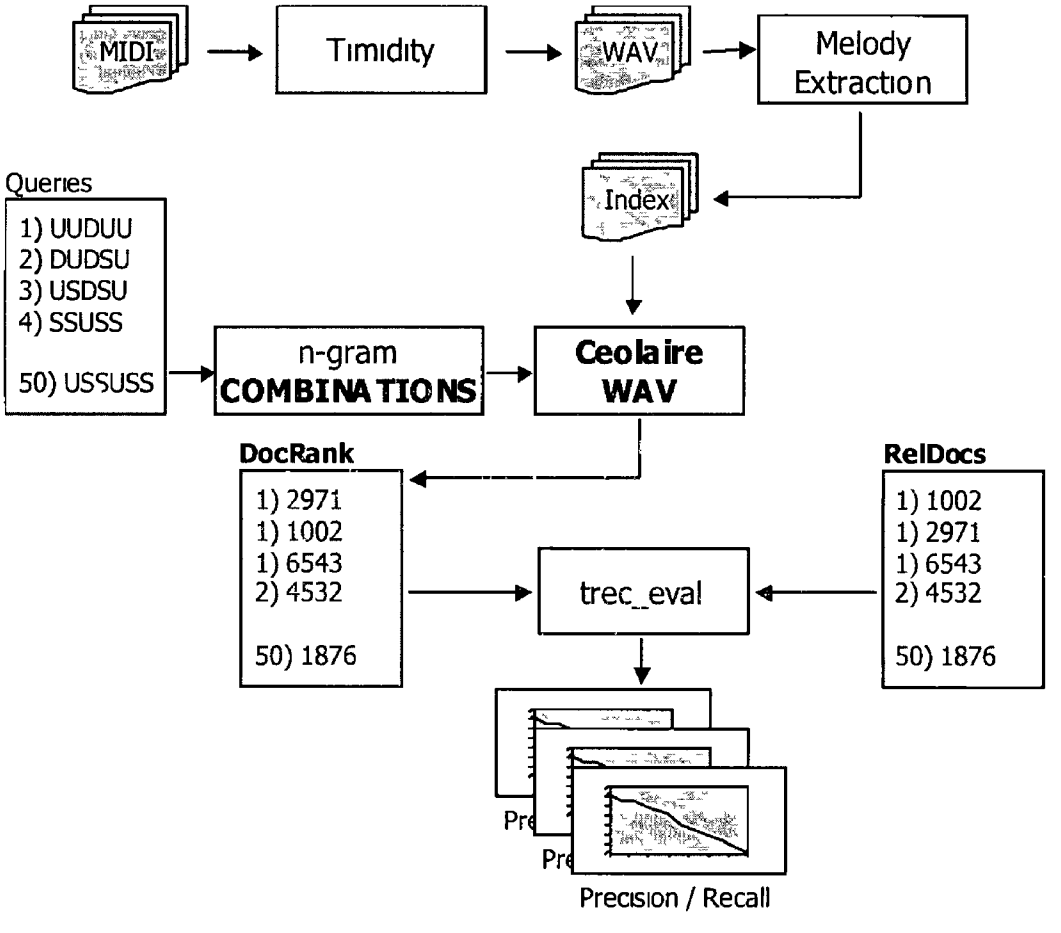


Figure 7 9 Steps undertaken to observe the retrieval performance of the CEOLAIRE WAV index using various combinations of n-grams

It is clear from the graphs that performance improves every time a larger n-gram is added This observation is repeated across all graphs For smaller groupings (2,3), there are higher concentrations of non-relevant documents, but as the groupings become larger (2,3,4,5,6,7,8), the relevant documents in the answer sets are ranked higher

Precision at 5, 10 and 20 documents for each retrieval run are listed in Table 7 4 and there is a relationship between the way the query is broken up into n-grams and the precision at various cut-off points Again, within each grouping we can observe how increasing the size of the n-gram helps to discriminate the relevant documents from the non-relevant documents

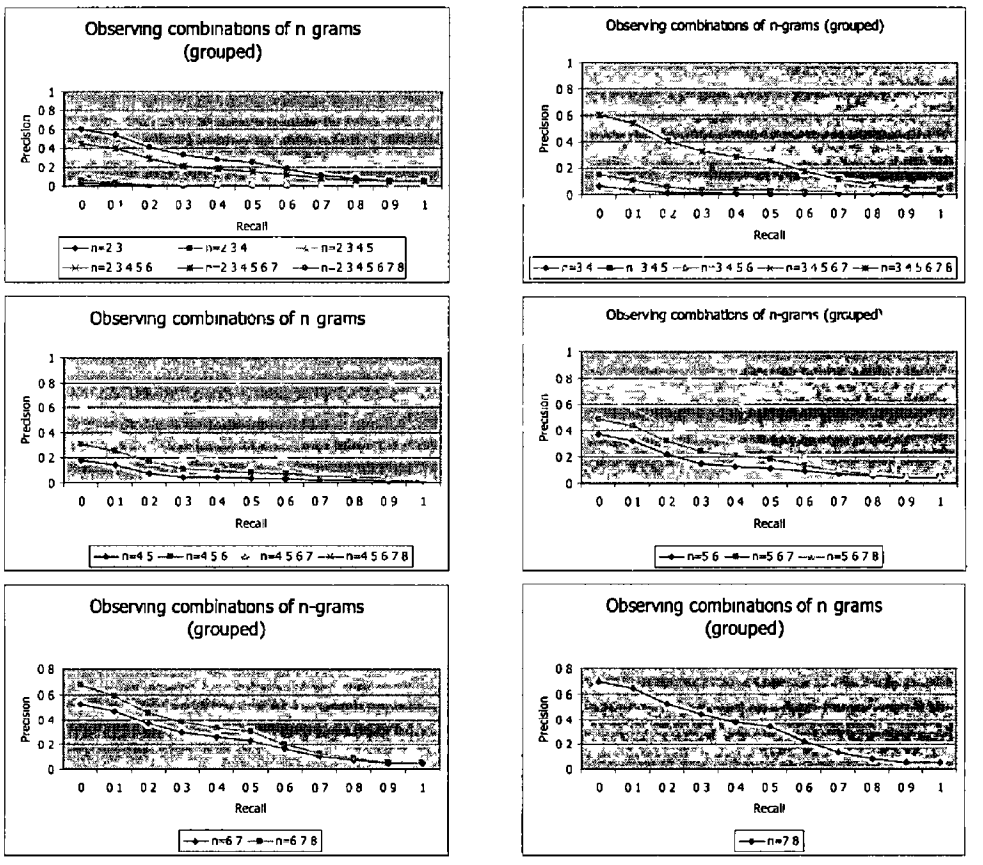


Figure 7 10 Observing the effect on precision and recall by combining various values of n

These groupings are shown in five graphs in Figure 7 11 and again, we can observe that for each of these groupings retrieval performance increases as larger sized n-grams are introduced to the group

The results show that the best retrieval performance is achieved with n-grams

RunID	Precision at	5	10	20
1	n=2,3	0 0120	0 0100	0 0060
7	n=2,3,4	0 0120	0 0160	0 0140
12	n=2,3,4,5	0 0320	0 0360	0 0280
12	n=2,3,4,5	0 0320	0 0360	0 0280
16	n=2,3,4,5,6	0 1000	0 0800	0 0733
19	**n=2,3,4,5,6,7**	**0 1800**	**0 1520**	**0 1270**
2	n=3,4	0 0160	0 0180	0 0130
8	n=3,4,5	0 0400	0 0380	0 0280
13	n=3,4,5,6	0 1000	0 8200	0 0700
17	n=3,4,5,6,7	0 1760	0 1540	0 1290
20	**n=3,4,5,6,7,8**	**0 2520**	**0 2320**	**0 1930**
3	n=4,5	0 0440	0 0420	0 0350
9	n=4,5,6	0 1040	0 880	0 0740
14	n=4,5,6,7	0 1800	0 1560	0 1350
18	**n=4,5,6,7,8**	**0 2560**	**0 2360**	**0 1950**
4	n=5,6	0 1240	0 0980	0 0880
10	n=5,6,7	0 1920	0 1600	0 1540
15	**n=5,6,7,8**	**0 2600**	**0 2680**	**0 2010**
5	n=6,7	0 2120	0 2000	0 1760
11	**n=6,7,8**	**0 3040**	**0 2840**	**0 2080**
6	**n=7,8**	**0 3640**	**0 3140**	**0 2300**

Table 7 4 Precision at 5, 10 and 20 documents when observing the performance of the CEOLAIRE WAV index using various combined values of n

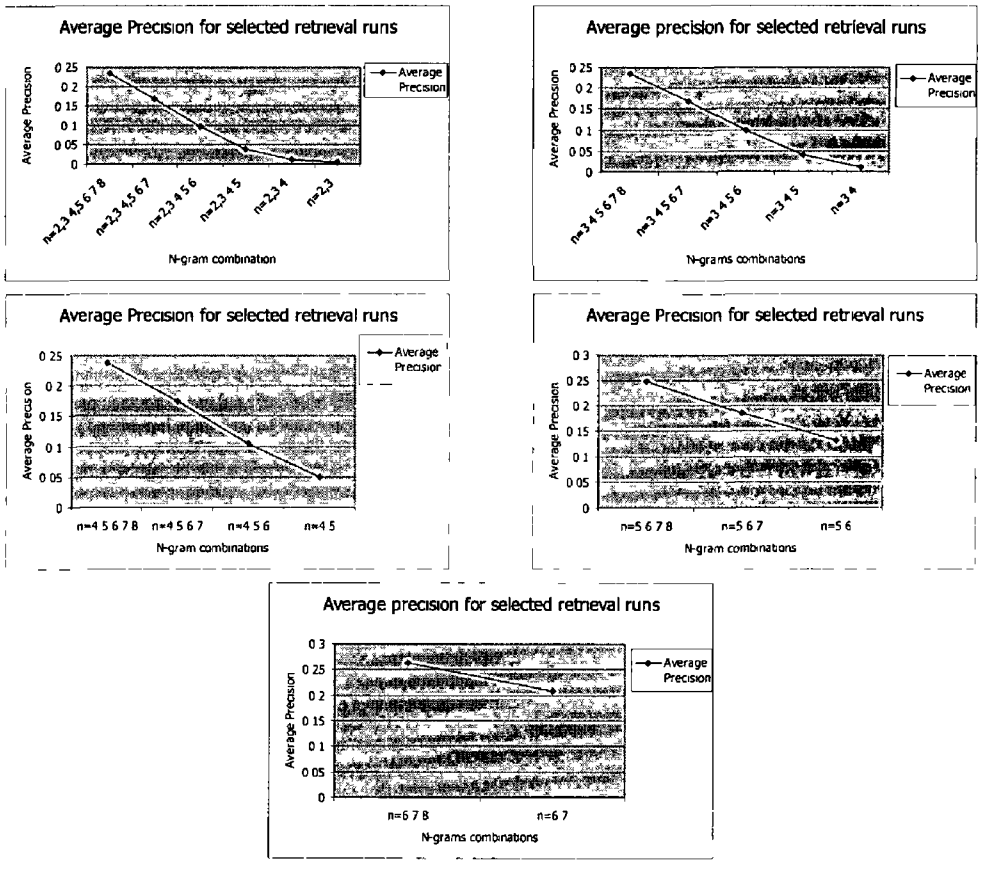


Figure 7 11 Average precision for each retrieval run

of size 7 and 8, which mirrors the performance from the second experiment The second best retrieval performance was observed using the n-gram combination 6, 7 and 8 and so on In answer to our original question, whether combining n-grams leads to an increase in retrieval effectiveness, the answer is "No" Rather, the impact is that retrieval effectiveness decreases every time a smaller sized n-gram is added to the retrieval run The reason for this is that new non-relevant documents are introduced into the result set and the relevant documents are ranked lower

Figure 7 12 plots the best retrieval run from the combined experiment against the best result from the single n-gram retrieval runs, n=8 Here, we can observe that the combined retrieval performance results lie below the best single retrieval performance of n=8, a slight decrease occurs every time a smaller valued n-gram is added We believe this to be as a result of smaller sized n-grams leading to an increase in the number of non-relevant documents in the answer set which has a negative effect on precision by forcing the relevant documents to be ranked lower than they should

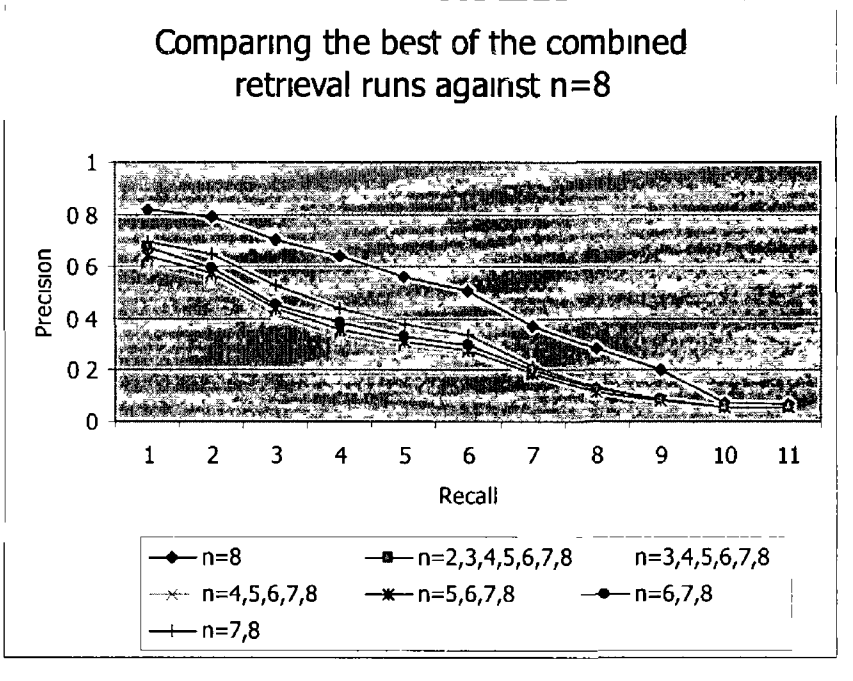


Figure 7 12 Comparing combined retrieval runs to single retrieval runs

7.4 Summary

In this chapter we presented the methodology and results of our evaluation of the effectiveness of CEOLAIRE as a music information retrieval system We undertook three experiments to evaluate CEOLAIRE

First, we observed the effectiveness of CEOLAIRE as a music information retrieval system by comparing its baseline performance against an index which we believe to be correct This experiment showed an average precision of 0 4941 The performance we obtained is due to using an index which is derived from our melody extraction process which we know experiences difficulties when extracting repeated notes at note boundaries

Next, we observed the systems retrieval effectiveness as an approximate matching system This experiment was undertaken by segmenting the query into n-grams of varying length, from 2 to 8 In total, seven retrieval runs were carried out and the results show poor retrieval performance when employing the use of smaller sized n-grams This is due to the introduction of non-relevant documents which cause the relevant documents to be ranked lower

The final experiment was carried out in order to discover what effect, if any, combining n-gram sizes has on retrieval effectiveness and whether there is a particular n-gram size combination which can lead to an improvement in retrieval effectiveness In comparison to the single valued n-gram experiment, we observed that combing n-gram sizes decreases retrieval performance every time a new smaller sized n-gram is added Again, this is due to non-relevant documents entering the answer set, causing the relevant documents to be ranked lower Combining n-grams does not improve retrieval effectiveness

We also observe that CEOLAIRE could have different levels of approximate matching by employing the use of the various sized n-grams The original trec_eval output from all experiments are presented in Appendix B

Chapter 8

Conclusion

8.1 Summary of dissertation

We started this dissertation by looking at the term "*Information Retrieval*" and examining the components required to build a basic information retrieval system We discussed the four basic components of an information retrieval system being gathering, indexing, searching and management As part of this process we reviewed some of the more conventional approaches applied to both indexing and retrieval including the use of **index files** and signature files and the more popular models used to resolve a user's information need expressed as a query to a retrieval system including the Boolean model, the vector space model and the probabilistic model and how techniques such as the vector space model and the probabilistic model can be employed to achieve ranking of answer sets We concluded our discussion of information retrieval with a explanation of how precision and recall figures can be used to indicate the performance effectiveness of an information retrieval system

Next, we took a look at the **basic properties of sound** and how they relate to hearing and the associated implications this has for the storage and manipulation of digital audio We observed how audio, first, has to be sampled in order to be digitally represented and we looked at some of the issues governing the sampling process including the Nyqvist theorem, bit resolution and aliasing, factors which all have to be taken into account when we want to record audio After that, we looked at **storage costs** associated with the high bit rates needed for raw audio

and observed how **compression algorithms** have evolved to reduce these bit rates, MP3 for example reducing a 1.4 Mbps audio file down to 128 kbps with virtually no loss in its perceived sound quality. We also observed that music sharing has become a consumer driven reality, posing challenges for the record industry which remain unresolved. We discussed the multitude of **formats** available for storing music, in raw, compressed and notation form and in what capacity each one should be used. This section concluded with a discussion on the importance of **digital music streaming** with a look at the more popular Internet multimedia streaming clients.

Once we had defined the basic properties of sound and looked at issues surrounding its manipulation, we were able to take an in depth analysis of techniques providing access to digital audio from a content point of view, particularly, the different methods of gaining access to the underlying structure of digital audio. We began this discussion by exploring the differences between the time domain and frequency domain followed by an introduction to different methods of gaining access to musical content from digital music sources using **frequency spectra** generation and analysis. We also explored some of the issues relating to their generation, for instance how signals have to be windowed before they can be reliably analysed. We took an in depth review of how **Fourier transforms** are derived and implemented. Following that we looked at how music is played by tuning instruments to a musical system known as the **equal tempering** scale and the derivation of values for that scale. We also looked at how music can be **represented as strings** using both Interval and Parsons representations and how music retrieval can be reduced to the task of **string matching** using exact or approximate string matching techniques.

Having defined and explained a lot of the issues surrounding music information retrieval, we undertook a **literature review**, reviewing the work by other researchers of music information retrieval and some of the systems which have been implemented, both systems that work with MIDI and some of the more recent attempts to implement systems working on raw polyphonic musical sources. From this literature review we observed that the current trend is towards building systems which can successfully negotiate features or structures from both monophonic

and polyphonic sources In the last few years the field of music information retrieval has been extended by incorporating the use of raw or compressed music files as a basis for experimentation, although the use of MIDI as a format for music information retrieval is of particular use in testing proof of concepts for both polyphonic and monophonic music sources as its structure is easily extracted

Next, we reviewed the system which this dissertation is based upon, CEOLAIRE We built this system in order to test the retrieval effectiveness of a music information retrieval system which is built upon conventional information retrieval techniques The system fits the profile of an information retrieval system described in the first chapter and employs the use of the vector space model for retrieval We explored the core components of CEOLAIRE, **melody extraction**, the **search engine** and the **user interface** We used fast Fourier transforms to generate frequency spectra which we then analysed for note information Once notes were discovered, we analysed the notes for **Interval** and **Parsons** in order to generate indexable documents for the search engine We discussed some of the **important factors** regarding the music collection which CEOLAIRE employs Importantly, we saw most note changes occur within an interval of 5 The longest musical fragment lasts 84 seconds while the shortest lasts 5 5 seconds Therefore, for retrieval we employ the use of a tf*idf retrieval algorithm incorporating document length normalisation We also observed how a piece of rendered manuscript music would be indexed by CEOLAIRE Metadata is stored in a database and document descriptors are used at retrieval time to access this data for presentation within the answer set This section concluded with a look at the user interface, introducing the different components and how they can be used to help combat the introduction of errors in a user's query From an end-user's perspective, Interval matching is provided for a user with a strong knowledge of music and who has the ability to formulate musical queries with a modest amount of certainty, while Parsons matching allows the less musically inclined user undertake melody retrieval We also provide both exact and approximate matching

Having presented the theory behind music information retrieval and a description of CEOLAIRE which is based on these theories, we then proceeded to evaluate various components of the system The first component to come under scrutiny is

the **extraction engine**, which is at the heart of our system An **evaluation** of any music information retrieval system must start with the process that generates the index We then presented a series of experiments which measured the effectiveness of the CEOLAIRE system's melody extraction engine These experiments were undertaken to observe the **effectiveness** of CEOLAIRE'S melody extraction from raw monophonic music files The effectiveness level of the melody extraction engine was measured by first creating a **reference collection** of structures representing music which we knew to be correct against which we compared structures derived automatically as a result of our **melody extraction process** Four experiments were undertaken in total The first experiment tested for exact matches between the two sets of structures The results we obtained showed poor performance which is due to aligning the note durations and some notes not being identified correctly The second experiment questioned whether reducing the note identifiers down to one octave would make a difference to performance A slight increase was observed and this is due to an anomaly in the extraction engine where in a few cases when a note decayed, it was identified as being an octave higher The third experiment observed note onsets ignoring duration and the results showed that on average, CEOLAIRE'S extraction engine was correct 97% of the time Next we observed the effectiveness of using Parsons to represent music and we observed that there was no increase in accuracy when compared to the note onsets experiment Overall, the results from these experiments show that CEOLAIRE'S extraction process is appropriate for use in the generation of an index

The music collection in the WAV format requires 27 GB which is excessive and we have seen how we can stream MP3 files so we decided to evaluate the use of MP3 files as a format suitable for use by CEOLAIRE We needed to investigate whether CEOLAIRE could effectively extract melodic information from MP3 files To achieve this, the collection of music files was encoded into MP3 and then decoded back into WAV at the following bit rates 128 kbps, 64 kbps and 32 kbps The experiments carried out earlier were then repeated here and we observed that MP3 files encoded at 128 kbps and 64 kbps achieved acceptable performance but files encoded at 32 kbps suffered decreased performance

It is obvious that when developing an extraction engine for music an evaluation must be carried out to allow it to be refined and improved upon

Finally, we evaluated the **effectiveness** of CEOLAIRE as a **music information retrieval system** CEOLAIRE is a music information retrieval system which derives its index automatically from WAV files and as such, this index can contain errors We identified that these errors are partially due to the difficulty of observing repeated notes We have seen that CEOLAIRE experiences difficulties when deriving Interval and Parsons structures from raw monophonic data The final set of experiments was undertaken in order to observe how these problems affect retrieval effectiveness

Other systems derive their music content describing structures directly from MIDI files and as such, it is trivial to build a music index These systems do not have to deal with problems associated with raw music We do not undertake a user evaluation because the evaluation of human ability to formulate music queries is beyond the scope of this dissertation Rather, we are measuring the effectiveness of CEOLAIRE as a music information retrieval system If we were to undertake a user evaluation we would further develop the interface along the lines of a query-by-example system Other interfaces could employ the use of query-by-humming, whistling etc or even employ the use of a simple text box which a user could manually enter Parsons or Interval notation Another interface which could be developed would work on top of Ceolaire's and would have a query-by-example component where the Interface aides the user by 'auto-completing' the query for the user, eliminating the need to recall the entire melody We would test these interfaces against baseline retrieval effectiveness in order to observe what effect if any various interfaces have on retrieval effectiveness

The first step in this experiment involved the generation of two musical corpora One is derived directly from MIDI files while the other is derived from our melody extraction process Next, 50 musical queries were generated and transcribed into Parsons before being submitted to CEOLAIRE to generate a list of documents which we knew to be correct This was done against the reference collection derived from MIDI files

The first experiment observed the **effectiveness** of CEOLAIRE as a music infor-

mation retrieval system by submitting the 50 queries to the CEOLAIRE WAV index and generating Precision / Recall values This showed that CEOLAIRE has an average precision of 0 4941 The performance we obtained is due to employing an index which is derived from our melody extraction process which we know experiences difficulties when extracting repeated notes at note boundaries It is worth noting that a small error in the extraction process can have a strong effect on retrieval performance which embeds itself as localised errors within the Parsons notation

Next, we observed the systems retrieval effectiveness as an **approximate matching system** This experiment was undertaken by segmenting the query into n-grams of varying length from 2 to 8 In total, seven retrieval runs were carried out and the results show poor retrieval performance when employing the use of smaller sized n-grams We believe that this is due to the introduction of non-relevant documents which cause the relevant documents to be ranked lower

The final experiment was carried out in order to discover what effect, if any, **combining n-gram sizes** has on retrieval and whether there is a particular n-gram size combination which gives improved retrieval effectiveness In comparison to the single valued n-gram experiment, we observed that combing n-gram sizes decreases retrieval performance every time a new smaller sized n-gram is added Again, this is due to non-relevant documents entering the answer set, causing the relevant documents to be ranked lower Combining n-grams does not improve retrieval effectiveness

The contributions we have made in this thesis are based around the music information retrieval system we built Our system allows for the indexing and subsequent retrieval of raw digital monophonic music sources We have taken a step away from using the MIDI format as source for music data structures, providing a retrieval framework which allows for searching of raw digital audio by first converting it to string representations We have shown how effective and efficient this type of retrieval can be through the evaluations we undertook No other researchers, to our knowledge, have evaluated a music information retrieval system this way Evaluation of music information retrieval systems is an issue which has been highlighted by the music retrieval community, notably by [Downie, 00] We believe that the

framework we chose to use could be used by other researchers when observing the effectiveness of their music retrieval systems

8.2 Future Work

We built a music information retrieval system which indexes raw monophonic music sources This is the first step to building a system which supports complex polyphonic music sources We believe that further development of the extraction engine could lead to acceptable performance on raw polyphonic instrumental music Doraisamy [Doraisamy, 01] developed a framework for indexing polyphonic music taken from MIDI files using the vector space model and this kind of model could be incorporated into CEOLAIRE in the future Monophonic music, as used in this thesis, is somewhat artificial in terms of what today's music industry produces, in that it is like playing a song on a piano with only one finger Modern music is naturally polyphonic and multi-instrumental and can contain sound effects and background drum beats, which make meaningful analysis a lot more difficult from an information retrieval perspective Our work and the work of the music information retrieval field is currently a long way from being able to index such music

The next step in our future work will be to review the extraction process in order to improve its ability to extract notes The first step of this process would be to build another layer on top of the extraction engine which analyses the power dissipation of musical notes as they are played This would make it easier to detect repeated notes Another approach which could be undertaken would be to not index repeated notes This would entail the note changes being recorded as either up (U) or down (D) This is an approach we would rather not pursue as it incurs a further loss of resolution which would have a negative effect on precision It may also be worthwhile undertaking an experiment to see what effect, if any, weighting the repeated notes lower at retrieval time has

As we develop CEOLAIRE to work with polyphonic music, employing the use of the MP3 file format may present problems as some frequency components may disappear altogether from the frequency domain, especially at low bit rates Therefore

as CEOLAIRE'S extraction engine is extended it may be necessary to test whether or not using the MP3 file format is problematic, especially at low bit rates of polyphonic music

Further work also includes developing the interface to help users generate queries CEOLAIRE'S current interface is intuitive and easy to use but it could be extended to a query-by-example system which could employ the use of a Hidden Markov Model to auto-complete a query for a user or to suggest alternative queries based on a few notes generated by the user Another idea would be to build a query-by-whistling, query-by-hummmg or query-by-something which takes a query in the WAV format and analyses the pitch contour This pitch contour could then be represented on CEOLAIRE'S query screen, where the user could review the query before its submission to the search engine The results from these interfaces could then be evaluated against the baseline results in chapter 7

Finally, yet another idea for future work is based around beat analysis The idea behind beat analysis is to discover drum-like instruments in polyphonic music in the frequency domain to build up a picture of the overall beat structure of a song The main use for this would be to discover songs with similar beats, rave music being the prime example as it normally has a very fast beat throughout the song

As in all research work within information retrieval we would like to compare the performance of our system against other retrieval systems although, currently, there is no existing framework which allows this to happen and there seems to be little incentive or drive from within the music information retrieval community to do this This is a pity as it means that progress in developing music information retrieval will be much slower than it could be

Bibliography

[1FM, 02] 1Fm, radio station (Internet and traditional broadcasts), available at URL http //www 1fm no/ (Last visited - July 2002)

[Adamson et al , 84] Adamson, G , Boreham, J The Use of an Association Measure Based on Character Structure to Identify Semantically Related Pairs of Words and Document Titles, *Information Storage and Retrieval*, #10, pages 253-260, 1984

[Agosti et al , 00] Agosti, M , Melucci, M Information Retrieval on the Web *In Proceedings of the 3rd European Summer School in Information Retrieval*, Italy, 2000

[Alltheweb, 02] Alltheweb, search engine, available at URL http //www alltheweb com/ (Last visited - July 2002)

[Altavista, 02] Altavista, search engine, available at URL http //www altavista com/ (Last visited - July 2002)

[Angell et al , 83] Angell, R , Freund, G , Willet, P , Automatic Spelling Correction Using a Trigram Similarity Measure, *Information Processing and Management*, 19(4) 255-261, 1983

[Atal et al , 84] Atal, B S , Schroeder R , Code-excited linear prediction (CELP) High-quality speech at very low bit rates, *In Proceedings of IEEE-ICASSP*, Tampa, FL, USA, pages 937-940, 1985

[Baeza-Yates et al , 99a] Baeza-Yates, R , Navarro, G Faster Approximate String Matching, In Algorithmica, 23(2) 127-158, 1999

[Baeza-Yates et al , 99b] Baeza-Yates, R Modern Information Retrieval, Addison Wesley, 1999

[BBCOGG, 02] BBC, testing Ogg audio streams, available at URL http //support bbc co uk/ogg/ (Last visited - July 2002)

[Beeferman, 97] Beeferman, D , QPD Query by Pitch Dynamics, Indexing Tonal Music By Content, 15-829 Course Project Available at URL http //www link cs cmu edu/, December 1997

[Berry et al , 99] Berry, M , Browne, M Understanding search engines Mathematical Modelling and Text Retrieval, SIAM, 1999

[Boyer et al , 77] Boyer, R S , A Fast String Searching Algorithm, *Communications of the ACM*, 20(10) 762-772, 1977

[BSD, 02] BSD license, available at URL http //www opensource org/licenses/bsd-license html (Last visited - July 2002)

[Cooley et al , 65] Cooley, J W and Tukey, J W , An algorithm for the machine calculation of complex Fourier series, *Mathematics of Computation*, 19(90) 297-301, 1965

[Damashek, 95] Damashek, M , Gauging Similarity with n-grams Language Independent Categorization of Text, *Science*, 267(5199) 843-848, 1995

[Doraisamy, 01] Doraisamy, S , An Approach Towards A Polyphonic Music Retrieval System, *In Proceedings of the 2nd ISMIR*, Bloomington, Indiana, USA, pages 145-165, 15-17 October, 2001

[Dovey, 01] Dovey, M , A technique for 'regular expression' style searching in polyphonic music, *In Proceedings of the 2nd ISMIR*, Bloomington, Indiana, USA, pages 179-185, 15 - 17 October, 2001

[Dowling, 78] Dowling, W J , Scale and contour Two components of a theory of memory for melodies, *Psychological Review,* 85(4) 341-354, 1978

[Downie, 00] Downie, S , Evaluation of a Simple and Effective Music Information Retrieval Method *In Proceedings ACM SIGIR Conference*, pages 73-80, 2000

[Foote, 99a] Foote, J , An overview of audio information retrieval, *In Multimedia Systems,* 7(1) 2-11, 1999

[Foote, 99b] Foote, J , Methods for the Automatic Analysis of Music and Audio, *FXPAL Technical Report TR-99-038,* December 1999

[Foote, 00] Foote J , ARTHUR Retrieving Orchestral Music by Long-Term Structure, *In Proceedings of the 1st ISMIR*, Plymouth, Massachusetts, USA, 23-25 October, 2000

[Ghias et al , 97] Ghias, A , Logan, J , Chamberlain, D and Smith, B Query by Humming - Musical Information Retrieval in an Audio Database, *ACM Multimedia* 95

[Glenbrook, 02] Glenbrook physics tutorial, available at URL http //www glenbrook k12 il us/gbssci/phys/Class/sound/u11l2d h (Last visited - July 2002)

[Guido, 02] Guido music information retrieval system, available at URL http //www informatik tu-darmstadt de/AFS/GUIDO/ and http //www noteserver org (Last visited - July 2002)

[Halıday et al , 97] Fundamentals of Phÿsıcs, Sıxth Edıtıon, Wıley, 1997

[Hammıng, 50] Hammıng, R W , Error-detectıng and error-correctıng codes, *Bell Sys Tech J* 29, 147160, 1950

[Harrıs, 78] Harrıs, F J On the Use of Wındows for Harmonıc Analysıs wıth Dıscrete Fourıer Transform, *In Proceedıngs of the IEEE*, 66(1) 51-83, January 1978

[Hoos et al , 01] Hoos, H , Renz, K , Gorg, T , GUIDO/MIR - an Experımental Musıcal Informatıon Retrıeval System based on GUIDO Musıc notatıon, *In Proceedıngs of the 2nd ISMIR*, Bloomıngton, Indıana, USA, pages 41-50, 15-17 October, 2001

[Huffman, 52] Huffman, D A , A Method for the Constructıon of Mınımum Redundancy Codes, *In Proceedıngs of the Instıtute of Radıo Engıneers,* 40(9) 1098-1101, September 1952

[ITU, 02] ITU G 726 (12/90), 40, 32, 24, 16 kbıt/s adaptıve dıfferentıal pulse code modulatıon (ADPCM) G 727 5-, 4-, 3- and 2-Bıts Sample Embedded Adaptıve Dıfferentıal Pulse Code Modulatıon (ADPCM), avaılable at URL http //www ıtu ınt/

[Kıentzle, 98] Kıentzle, T , A Programmers Guıde to Sound Addıson Wesley, 1998

[Knuth et al , 77] Knuth, D E , Morrıs(Jr), J H , Pratt, V R , Fast pattern matchıng ın strıngs, *SIAM Journal on Computıng*, 6(1) 323-350, 1977

[Kornstadt, 98] Kornstadt, A Themefinder A Web-based Melodıc Search Tool, Melodıc Comparıson Concepts, Procedures and Applıcatıons, *Computıng ın Musıcology II*, pages 231-236, 1998

[Lee, 97] Lee, J H , Analysis of multiple evidence combination *In Proceedings of ACM SIGIR Conference*, Philadelphia, USA, pages 267-276, July 1997

[Lempel et al , 77] Lempel, A , Ziv, J , A universal algorithm for sequential data compression *IEEE Transactions on Information Theory*, pages 337-343, May 1977

[Lemstrom et al , 00] Lemstrom, K , Perttu S , SEMEX - An efficient Music Retrieval Prototype *In Proceedings of the 1st ISMIR*, Plymouth, Massachusetts, USA, 23-25 October, 2000

[1] Levenshtein, V Binary Codes Capable of Correcting Deletions, Insertions and Reversals, Soviet Phys Dokl, 10 707-710, 1966

[McIlroy, 82] McIlroy, M D , Development of a Spelling List, *IEEE Transaction on Communications*, Com-30(1) 91-99, 1982

[McNab et al , 97] McNab, R , Smith, L Bambridge, D and Witten, I , The New Zealand Digital Library Melody Index, *D-Lib Magazine*, also available at URL http //www dlib org/dlib/may97/meldex/05witten html, May 1997

[Melodyhound, 02] Melodyhound, music information retrieval system, available at URL http //www melodyhound com/ (Last visited - July 2002)

[Minradio, 02] Minradio, Internet based radio station, available at URL http //www minradio com/ (Last visited - July 2002)

[Morris et al , 75] Morris, R , Cherry, L Computer Detection of Typographical Errors, *IEEE Transactions on Professional Communications*, 18(1) 54-56, March 1975

[mpg123, 02] mpg123, MP3 decoder, available at URL http //www mpg123 de/ (Last visited - July 2002)

[Myers, 98] Myers, G A fast bit-vector algorithm for approximate string matching based on dynamic programming, *In Proceedings of Combinatorial Pattern Matching*, Piscataway, N J , pages 113

[Napster, 02] Napster, peer-to-peer file sharing service, available at URL http //www napster com/ (Last visited - July 2002)

[Ogg, 02] Ogg library, available at URL http //www vorbis com/ (Last visited - July 2002)

[P4, 02] P4, radio station (Internet and traditional broadcasts), available at URL http //www p4 no/ (Last visited - July 2002)

[Philips, 01] Philips, Trying to name that tune? available at URL http //www philips com/InformationCenter/Global/FArticleI (Last visited - July 2002)

[Pan, 95] Pan, D A Tutorial on MPEG/Audio Compression, *IEEE Multimedia*, 2(2) 60-74, Summer 1995

[Peterson, 80] Peterson, J L , Computer Programs for Detecting and Correcting Spelling Errors, *Communications of the ACM*, 23(12) 676-687, 1980

[Plumbley, 01] Plumbley, M D , Abdallah, S A , Bello, J P , Davies, M E , Klingseisen, J , Monti, G and Sandler, M B ICA and Related Models Applied to Audio Analysis and Separation *Fourth International ICSC Symposium on Soft Computing and Intelligent Systems for Industry*, Paisley, Scotland, United Kingdom, 26-29 June, 2001

[Porter, 80] Porter, M F An algorithm for suffix stripping, *Program*, 14(3) 130-137, 1980

[Pratt et al , 42] Pratt, Fletcher, Secret and Urgent, Blue Ribbon Books, Garden City, N J , Chapter 4, 1942

[Press et al , 92] Press, W H , Flannery, B P , Teukolsky, S A , Vetterling, W T Wavelet Transforms, *§13 10 in Numerical Recipes in FORTRAN The Art of Scientific Computing*, 2nd ed Cambridge, England Cambridge University Press, pages 584-599, 1992

[Pye, 00] Pye, D , Content-Based Methods for the Management of Digital Music, *In Proceedings of the ICASSP*, 2000

[van Rijsbergen, 79] van Rijsbergen, C J , *Information Retrieval*, 2nd Edition, Butterworths, London pages 114-115, 1979

[Robertson et al , 76] Robertson, S E , Spark Jones, K Relevance Weighting of Search Terms, *American Society for Information Science*, 27(3) 57-71, 1976

[Salton, 75] Salton, G Dynamic Information and Library Processing, Prentice-Hall Inc , Eaglewod, New Jersey, 1975

[Smeaton, 98] Smeaton A F , Independence of Contributing Retrieval Strategies in Data Fusion for Effective Information Retrieval, *In Proceedings of BCS-IRSG Colloquium on Information Retrieval*, 1998

[Smeaton et al , 98] Smeaton, A F , Morony, M , Quinn, G and Scaife, R , Taiscealai Information Retrieval from an Archive of Spoken Radio News, *In Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries (ECDL '98)*, Heraklion, Crete, pages 429-442, 21-23 September 1998

[Smeaton et al , 01] Smeaton, A F , Murphy, N , O'Connor, N E , Marlow, S , Lee, H , McDonald, K , Browne, P and Ye, J , The Fischlar Digital Video System A Digital Library of Broadcast TV Programmes, *In Proceedings of the Joint ACM IEEE Conference on Digital Libraries*, Roanoake, North Carolina, June 2001

[Liebchen et al , 97] Liebchen, T , Purat, M , Noll, P Lossless Transform Coding of Audio Signals *2nd AES Convention*, Munich, 1997

[sox, 02] Sox, Sound eXchange universal sound sample translator, available at URL http //home sprynet com/~cbagwell/sox html (Last visited - July 2002)

[Starcd, 02] StarCD online music purchasing system, available at URL http //www starcd com/ (Last visited - July 2002)

[TCLEX, 95] Transnational College of LEX, Translated from Japanese by Alan Gleason Who Is Fourier? A Mathematical Adventure

[Makhoul, 75] Makhoul, J , Linear prediction a tutorial review, *In Proceedings of the IEEE*, 63(4) 561-80, 1975

[Timidity, 02] Timidity Synthesiser MIDI to WAVE converter/player Available at URL http //www goice co jp/member/mo/timidity/ (Last visited - July 2002)

[Tague-Sutcliffe, 92] Tague-Sutcliffe, J The pragmatics of information retrieval experimentation, revisited, *Information Processing and Management*, 28(4) 467-490, 1992

[TREC, 02] TREC, Text **RE**trieval **C**onference, available at URL http //trec nist gov/ (Last visited - July 2002)

[Themefinder, 02] Themefinder, music information retrieval system, available at URL http //www themefinder com/ (Last visited - July 2002)

[Tucker et al , 99] Tucker, R , Robinson, T , Christie, J , Seymour, C , Recognition-Compatible Speech Compression for Stored Speech, *In Proceedings of the ESCA workshop Accessing information in spoken audio*, pages 69-72, April 1999, also available at URL http //svr-www eng cam ac uk/~ajr/esca99/ (Last visited- July 2002)

[Witten et al , 99] Witten, I H , Moffat, A , Bell, T C Managing Gigabytes, Morgan Kaufmann Publishing, 1999

[Wold et al , 96] Wold, E , Blum, T Keislar, D and Wheaton, J Content-Based Classification, Search, and Retrieval of Audio, *IEEE Multimedia*, 3(3) 27-36, 1996

[xaudio, 02] xaudio MP3 library, available at URL http //www xaudio com/ (Last visited- July 2002)

Appendix A

Queries

The queries and their Parsons equivalence used in the experiments undertaken as part of the search engine retrieval performance in chapter 6 are shown in Table A 1

ID	Query	Parsons	ID	Query	Parsons
1	CEEEGC'C'BG	usssuusdd	26	EGCECG,G,G,E,	uududdssd
2	GC'C'D'E'G'E'G'F'	usuuududu	27	GCBDACCDE	uududusuu
3	EEDCEGGGC'D'	sdduussuu	28	C'GADCCDBG	dduddsdudd
4	GGGECCCDDG	sssddssusd	29	FFCCCDDDCC	ssdssussds
5	BbDEbFEbBbDCCCEbF	uuudduds suu	30	EGCGBD'G'D'D'	uuuduuuds
6	A#C#BG#A#BA#C#D#D#D#	uudduudu duss	31	AGF#F#F#DDE	dddssdsu
7	AEGGGEACGEEE	ddussduuddss	32	DGBb'C'D'Eb'C'DbGG	duuuduuudds
8	FFAFACDAAGC	ssuduuudsdu	33	FGAGFDAACE	uuudddusuu
9	F#F#DEEF#F#DA	dsdususdd	34	GF#DEDDCEAG	uddudsduud
10	GC'GECCCG,G,	uudddssds	35	BbGFFEbDDEbF	dddsddsuu
11	AADF#DDF#AG	ssdudsuud	36	GACBC'D'BGAEDC	uuuduudduddd
12	EG#AEG#EBBA	uuuduuusu	37	FACAC'D'AGG	uuuduuddds
13	C'GGAFFFDGGG	ddsudssduuu	38	ECDECEFECC	dduuduudds
14	FFEDDCEGFF	sddsduuds	39	DDGGAAGF#F#	ssususdds
15	GCGGEG'GG'C'EG	sudsuuduudd	40	AAAG#G#AC'E	ussdsuud
16	ECABABCDAA	ddduduuuds	41	GGGC'GC'E'GG	dssuduuds
17	FAGFDFGCCC	uuddduudss	42	BBG#BEA#A#A#B	ssduudssu
18	GGEEDDEDDG	dsdsdudsu	43	GCCCAAAGD	sdssussdd
19	AC'C'AEEFACG#A	uusddsuuudu	44	AbAbAbEbEbEbDbDbDbEb	ussdssdsu
20	GEFEEDGBCBG	ddudsduuudd	45	FGFGAFGGGC	duduudussd
21	BD#F#ED#C#EF#D#BB	uuuddduudds	46	EG#EEC#G#ABB	uudsddsuus
22	GCEGEDGGGGD	dduuddusssd	47	AAAGGGFFA	dssdssdsu
23	AF#EDDDC#EC#A	udddssdudd	48	C'AFC'AAAGC	uddudssdd
24	CDCEFEDCCG	uuduudddsd	49	AFFFDECC	udssduds
25	AEDFGEDDDC	ddduuddssd	50	C'C'C'FGAGC'CC	ssduududs

Table A 1 Queries for experments

Appendix B

Evaluation Results

The Precision Recall values from the experiments are shown below

B.1 Experiment 1

Queryid (Num) 50

Total number of documents over all queries

Retrieved 511

Relevant 559

Rel_ret 351

Interpolated Recall - Precision Averages

at 0 00 0 9056

at 0 10 0 8822

at 0 20 0 8108

at 0 30 0 7435

at 0 40 0 6646

at 0 50 0 5966

at 0 60 0 4270

at 0 70 0 2917

at 0 80 0 2052

at 0 90 0 0800

at 1 00 0 0800

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 4941

Precision

At 5 docs 0 6400

At 10 docs 0 5100

At 15 docs 0 4133

At 20 docs 0 3310

At 30 docs 0 2273

At 100 docs 0 0702

At 200 docs 0 0351

At 500 docs 0 0140

At 1000 docs 0 0070

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 5755

B.2 Experiment 2

Results from n-gram size = 2

Queryid (Num) 50

Total number of documents over all queries

Retrieved 444338

Relevant 559

Rel_ret 539

Interpolated Recall - Precision Averages

at 0 00 0 0290

at 0 10 0 0163

at 0 20 0 0079

at 0 30 0 0047

at 0 40 0 0040

at 0 50 0 0035

at 0 60 0 0032

at 0 70 0 0027

at 0 80 0 0025

at 0 90 0 0022

at 1 00 0 0015

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0058

Precision

At 5 docs 0 0040

At 10 docs 0 0100

At 15 docs 0 0080

At 20 docs 0 0060

At 30 docs 0 0087

At 100 docs 0 0050

At 200 docs 0 0038

At 500 docs 0 0030

At 1000 docs 0 0025

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0074

Results from n-gram size = 3

Queryid (Num) 50

Total number of documents over all queries

Retrieved 433887

Relevant 559

Rel_ret 539

Interpolated Recall - Precision Averages

at 0 00 0 0622

at 0 10 0 0440

at 0 20 0 0202

at 0 30 0 0132

at 0 40 0 0103

at 0 50 0 0090

at 0 60 0 0079

at 0 70 0 0063

at 0 80 0 0052

at 0 90 0 0041

at 1 00 0 0029

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0138

Precision

At 5 docs 0 0160

At 10 docs 0 0200

At 15 docs 0 0173

At 20 docs 0 0160

At 30 docs 0 0133

At 100 docs 0 0106

At 200 docs 0 0098

At 500 docs 0 0070

At 1000 docs 0 0055

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0190

Results from n-gram size = 4

Queryid (Num) 50

Total number of documents over all queries

Retrieved 366588

Relevant 559

Rel_ret 537

Interpolated Recall - Precision Averages

at 0 00 0 1674

at 0 10 0 1322

at 0 20 0 0741

at 0 30 0 0507

at 0 40 0 0443

at 0 50 0 0413

at 0 60 0 0343

at 0 70 0 0282

at 0 80 0 0234

at 0 90 0 0187

at 1 00 0 0156

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0507

Precision

At 5 docs 0 0440

At 10 docs 0 0440

At 15 docs 0 0387

At 20 docs 0 0400

At 30 docs 0 0333

At 100 docs 0 0288

At 200 docs 0 0215

At 500 docs 0 0138

At 1000 docs 0 0085

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0524

Results from n-gram size = 5

Queryid (Num) 50

Total number of documents over all queries

Retrieved 226395

Relevant 559

Rel_ret 516

Interpolated Recall - Precision Averages

at 0 00 0 3993

at 0 10 0 3504

at 0 20 0 2285

at 0 30 0 1573

at 0 40 0 1380

at 0 50 0 1274

at 0 60 0 1089

at 0 70 0 0869

at 0 80 0 0756

at 0 90 0 0520

at 1 00 0 0468

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 1421

Precision

At 5 docs 0 1360

At 10 docs 0 1100

At 15 docs 0 0987

At 20 docs 0 0940

At 30 docs 0 0827

At 100 docs 0 0532

At 200 docs 0 0343

At 500 docs 0 0168

At 1000 docs 0 0091

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 1450

Results from n-gram size = 6

Queryid (Num) 50

Total number of documents over all queries

Retrieved 99722

Relevant 559

Rel_ret 477

Interpolated Recall - Precision Averages

at 0 00 0 5418

at 0 10 0 4914

at 0 20 0 3727

at 0 30 0 2934

at 0 40 0 2762

at 0 50 0 2561

at 0 60 0 2043

at 0 70 0 1478

at 0 80 0 1233

at 0 90 0 0626

at 1 00 0 0541

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 2248

Precision

At 5 docs 0 2080

At 10 docs 0 1940

At 15 docs 0 1873

At 20 docs 0 1910

At 30 docs 0 1520

At 100 docs 0 0710

At 200 docs 0 0411

At 500 docs 0 0172

At 1000 docs 0 0091

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 2222

Results from n-gram size = 7

Queryid (Num) 50

Total number of documents over all queries

Retrieved 36460

Relevant 559

Rel_ret 445

Interpolated Recall - Precision Averages

at 0 00 0 7010

at 0 10 0 6466

at 0 20 0 5461

at 0 30 0 4566

at 0 40 0 3984

at 0 50 0 3695

at 0 60 0 2804

at 0 70 0 2162

at 0 80 0 1685

at 0 90 0 0729

at 1 00 0 0644

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 3228

Precision

At 5 docs 0 3560

At 10 docs 0 3280

At 15 docs 0 2907

At 20 docs 0 2540

At 30 docs 0 1920

At 100 docs 0 0782

At 200 docs 0 0409

At 500 docs 0 0170

At 1000 docs 0 0087

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 3566

Results from n-gram size = 8

Queryid (Num) 50

Total number of documents over all queries

Retrieved 11339

Relevant 559

Rel_ret 407

Interpolated Recall - Precision Averages

at 0 00 0 8184

at 0 10 0 7897

at 0 20 0 6997

at 0 30 0 6420

at 0 40 0 5584

at 0 50 0 5080

at 0 60 0 3705

at 0 70 0 2855

at 0 80 0 2034

at 0 90 0 0740

at 1 00 0 0695

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 4264

Precision

At 5 docs 0 5080

At 10 docs 0 4480

At 15 docs 0 3680

At 20 docs 0 3060

At 30 docs 0 2300

At 100 docs 0 0776

At 200 docs 0 0394

At 500 docs 0 0161

At 1000 docs 0 0081

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 4843

B.3 Experiment 3

Results from combined n-gram sizes = 2,3

Queryid (Num) 50

Total number of documents over all queries

Retrieved 405710

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 0326

at 0 10 0 0191

at 0 20 0 0053

at 0 30 0 0045

at 0 40 0 0040

at 0 50 0 0033

at 0 60 0 0029

at 0 70 0 0024

at 0 80 0 0021

at 0 90 0 0012

at 1 00 0 0008

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0056

Precision

At 5 docs 0 0120

At 10 docs 0 0100

At 15 docs 0 0067

At 20 docs 0 0060

At 30 docs 0 0067

At 100 docs 0 0044

At 200 docs 0 0037

At 500 docs 0 0026

At 1000 docs 0 0023

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0084

Results from combined n-gram sizes = 3,4

Queryid (Num) 50

Total number of documents over all queries

Retrieved 396086

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 0625

at 0 10 0 0365

at 0 20 0 0162

at 0 30 0 0109

at 0 40 0 0094

at 0 50 0 0084

at 0 60 0 0072

at 0 70 0 0054

at 0 80 0 0042

at 0 90 0 0023

at 1 00 0 0018

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0121

Precision

At 5 docs 0 0160

At 10 docs 0 0180

At 15 docs 0 0133

At 20 docs 0 0130

At 30 docs 0 0107

At 100 docs 0 0106

At 200 docs 0 0096

At 500 docs 0 0065

At 1000 docs 0 0052

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0151

Results from combined n-gram sizes = 4,5

Queryid (Num) 50

Total number of documents over all queries

Retrieved 334737

Relevant 559

Rel_ret 479

Interpolated Recall - Precision Averages

at 0 00 0 1790

at 0 10 0 1392

at 0 20 0 0728

at 0 30 0 0486

at 0 40 0 0442

at 0 50 0 0404

at 0 60 0 0328

at 0 70 0 0250

at 0 80 0 0202

at 0 90 0 0160

at 1 00 0 0154

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0503

Precision

At 5 docs 0 0440

At 10 docs 0 0420

At 15 docs 0 0360

At 20 docs 0 0350

At 30 docs 0 0307

At 100 docs 0 0264

At 200 docs 0 0202

At 500 docs 0 0124

At 1000 docs 0 0077

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0504

Results from combined n-gram sizes = 5,6

Queryid (Num) 50

Total number of documents over all queries

Retrieved 206337

Relevant 559

Rel_ret 460

Interpolated Recall - Precision Averages

at 0 00 0 3693

at 0 10 0 3191

at 0 20 0 2229

at 0 30 0 1519

at 0 40 0 1320

at 0 50 0 1148

at 0 60 0 0948

at 0 70 0 0702

at 0 80 0 0558

at 0 90 0 0446

at 1 00 0 0442

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 1312

Precision

At 5 docs 0 1240

At 10 docs 0 0980

At 15 docs 0 0973

At 20 docs 0 0880

At 30 docs 0 0827

At 100 docs 0 0484

At 200 docs 0 0314

At 500 docs 0 0150

At 1000 docs 0 0081

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 1375

Results from combined n-gram sizes = 6,7

Queryid (Num) 50

Total number of documents over all queries

Retrieved 90995

Relevant 559

Rel_ret 425

Interpolated Recall - Precision Averages

at 0 00 0 5311

at 0 10 0 4679

at 0 20 0 3634

at 0 30 0 2927

at 0 40 0 2588

at 0 50 0 2256

at 0 60 0 1604

at 0 70 0 1043

at 0 80 0 0761

at 0 90 0 0495

at 1 00 0 0489

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 2084

Precision

At 5 docs 0 2120

At 10 docs 0 2000

At 15 docs 0 1947

At 20 docs 0 1760

At 30 docs 0 1433

At 100 docs 0 0640

At 200 docs 0 0364

At 500 docs 0 0152

At 1000 docs 0 0081

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 2252

Results from combined n-gram sizes = 7,8

Queryid (Num) 50

Total number of documents over all queries

Retrieved 33212

Relevant 559

Rel_ret 394

Interpolated Recall - Precision Averages

at 0 00 0 6991

at 0 10 0 6459

at 0 20 0 5276

at 0 30 0 4414

at 0 40 0 3772

at 0 50 0 3357

at 0 60 0 2168

at 0 70 0 1338

at 0 80 0 0812

at 0 90 0 0531

at 1 00 0 0520

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 2944

Precision

At 5 docs 0 3640

At 10 docs 0 3140

At 15 docs 0 2693

At 20 docs 0 2300

At 30 docs 0 1767

At 100 docs 0 0700

At 200 docs 0 0360

At 500 docs 0 0150

At 1000 docs 0 0077

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 3402

Results from combined n-gram sizes = 2,3,4

Queryid (Num) 50

Total number of documents over all queries

Retrieved 405661

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 0621

at 0 10 0 0372

at 0 20 0 0145

at 0 30 0 0101

at 0 40 0 0086

at 0 50 0 0074

at 0 60 0 0062

at 0 70 0 0049

at 0 80 0 0038

at 0 90 0 0021

at 1 00 0 0016

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0115

Precision

At 5 docs 0 0120

At 10 docs 0 0160

At 15 docs 0 0133

At 20 docs 0 0140

At 30 docs 0 0100

At 100 docs 0 0098

At 200 docs 0 0088

At 500 docs 0 0062

At 1000 docs 0 0048

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0144

Results from combined n-gram sizes = 3,4,5

Queryid (Num) 50

Total number of documents over all queries

Retrieved 395931

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 1496

at 0 10 0 1056

at 0 20 0 0578

at 0 30 0 0384

at 0 40 0 0349

at 0 50 0 0322

at 0 60 0 0273

at 0 70 0 0213

at 0 80 0 0182

at 0 90 0 0140

at 1 00 0 0134

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0401

Precision

At 5 docs 0 0400

At 10 docs 0 0380

At 15 docs 0 0333

At 20 docs 0 0280

At 30 docs 0 0287

At 100 docs 0 0246

At 200 docs 0 0174

At 500 docs 0 0118

At 1000 docs 0 0076

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0372

Results from combined n-gram sizes = 4,5,6

Queryid (Num) 50

Total number of documents over all queries

Retrieved 334155

Relevant 559

Rel_ret 479

Interpolated Recall - Precision Averages

at 0 00 0 3118

at 0 10 0 2572

at 0 20 0 1766

at 0 30 0 1138

at 0 40 0 1005

at 0 50 0 0859

at 0 60 0 0750

at 0 70 0 0545

at 0 80 0 0429

at 0 90 0 0346

at 1 00 0 0339

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 1045

Precision

At 5 docs 0 1040

At 10 docs 0 0880

At 15 docs 0 0813

At 20 docs 0 0740

At 30 docs 0 0707

At 100 docs 0 0450

At 200 docs 0 0298

At 500 docs 0 0148

At 1000 docs 0 0081

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 1046

Results from combined n-gram sizes = 5,6,7

Queryid (Num) 50

Total number of documents over all queries

Retrieved 206033

Relevant 559

Rel_ret 460

Interpolated Recall - Precision Averages

at 0 00 0 4877

at 0 10 0 4340

at 0 20 0 3218

at 0 30 0 2453

at 0 40 0 2173

at 0 50 0 1875

at 0 60 0 1387

at 0 70 0 0935

at 0 80 0 0702

at 0 90 0 0481

at 1 00 0 0477

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 1860

Precision

At 5 docs 0 1920

At 10 docs 0 1600

At 15 docs 0 1573

At 20 docs 0 1540

At 30 docs 0 1273

At 100 docs 0 0602

At 200 docs 0 0351

At 500 docs 0 0155

At 1000 docs 0 0083

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 1900

Results from combined n-gram sizes = 6,7,8

Queryid (Num) 50

Total number of documents over all queries

Retrieved 90816

Relevant 559

Rel_ret 424

Interpolated Recall - Precision Averages

at 0 00 0 6759

at 0 10 0 5963

at 0 20 0 4543

at 0 30 0 3826

at 0 40 0 3320

at 0 50 0 2988

at 0 60 0 2014

at 0 70 0 1261

at 0 80 0 0853

at 0 90 0 0533

at 1 00 0 0523

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 2643

Precision

At 5 docs 0 3040

At 10 docs 0 2840

At 15 docs 0 2480

At 20 docs 0 2080

At 30 docs 0 1687

At 100 docs 0 0700

At 200 docs 0 0366

At 500 docs 0 0155

At 1000 docs 0 0082

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 3070

Results from combined n-gram sizes = 2,3,4,5

Queryid (Num) 50

Total number of documents over all queries

Retrieved 405640

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 1444

at 0 10 0 0991

at 0 20 0 0577

at 0 30 0 0374

at 0 40 0 0340

at 0 50 0 0314

at 0 60 0 0268

at 0 70 0 0209

at 0 80 0 0180

at 0 90 0 0139

at 1 00 0 0134

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0389

Precision

At 5 docs 0 0320

At 10 docs 0 0360

At 15 docs 0 0333

At 20 docs 0 0280

At 30 docs 0 0273

At 100 docs 0 0238

At 200 docs 0 0170

At 500 docs 0 0118

At 1000 docs 0 0075

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0372

Results from combined n-gram sizes = 3,4,5,6

Queryid (Num) 50

Total number of documents over all queries

Retrieved 395850

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 2924

at 0 10 0 2357

at 0 20 0 1606

at 0 30 0 1047

at 0 40 0 0923

at 0 50 0 0798

at 0 60 0 0705

at 0 70 0 0529

at 0 80 0 0426

at 0 90 0 0344

at 1 00 0 0337

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0980

Precision

At 5 docs 0 1000

At 10 docs 0 0820

At 15 docs 0 0760

At 20 docs 0 0700

At 30 docs 0 0653

At 100 docs 0 0446

At 200 docs 0 0287

At 500 docs 0 0145

At 1000 docs 0 0081

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0959

Results from combined n-gram sizes = 4,5,6,7

Queryid (Num) 50

Total number of documents over all queries

Retrieved 334076

Relevant 559

Rel_ret 479

Interpolated Recall - Precision Averages

at 0 00 0 4569

at 0 10 0 4127

at 0 20 0 2974

at 0 30 0 2189

at 0 40 0 1913

at 0 50 0 1636

at 0 60 0 1267

at 0 70 0 0865

at 0 80 0 0655

at 0 90 0 0481

at 1 00 0 0474

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 1733

Precision

At 5 docs 0 1800

At 10 docs 0 1560

At 15 docs 0 1440

At 20 docs 0 1350

At 30 docs 0 1213

At 100 docs 0 0590

At 200 docs 0 0344

At 500 docs 0 0154

At 1000 docs 0 0083

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 1904

Results from combined n-gram sizes = 5,6,7,8

Queryid (Num) 50

Total number of documents over all queries

Retrieved 205949

Relevant 559

Rel_ret 460

Interpolated Recall - Precision Averages

at 0 00 0 6438

at 0 10 0 5723

at 0 20 0 4359

at 0 30 0 3550

at 0 40 0 3101

at 0 50 0 2773

at 0 60 0 1896

at 0 70 0 1175

at 0 80 0 0813

at 0 90 0 0519

at 1 00 0 0510

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 2473

Precision

At 5 docs 0 2600

At 10 docs 0 2680

At 15 docs 0 2333

At 20 docs 0 2010

At 30 docs 0 1613

At 100 docs 0 0684

At 200 docs 0 0364

At 500 docs 0 0156

At 1000 docs 0 0083

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 2907

Results from combined n-gram sizes = 2,3,4,5

Queryid (Num) 50

Total number of documents over all queries

Retrieved 405626

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 2929

at 0 10 0 2340

at 0 20 0 1595

at 0 30 0 1034

at 0 40 0 0913

at 0 50 0 0790

at 0 60 0 0699

at 0 70 0 0527

at 0 80 0 0424

at 0 90 0 0344

at 1 00 0 0337

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 0975

Precision

At 5 docs 0 1000

At 10 docs 0 0800

At 15 docs 0 0733

At 20 docs 0 0700

At 30 docs 0 0647

At 100 docs 0 0444

At 200 docs 0 0288

At 500 docs 0 0145

At 1000 docs 0 0081

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 0948

Results from combined n-gram sizes = 3,4,5,6

Queryid (Num) 50

Total number of documents over all queries

Retrieved 395898

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 4486

at 0 10 0 3989

at 0 20 0 2899

at 0 30 0 2079

at 0 40 0 1803

at 0 50 0 1552

at 0 60 0 1214

at 0 70 0 0848

at 0 80 0 0652

at 0 90 0 0482

at 1 00 0 0475

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 1683

Precision

At 5 docs 0 1760

At 10 docs 0 1540

At 15 docs 0 1413

At 20 docs 0 1290

At 30 docs 0 1193

At 100 docs 0 0584

At 200 docs 0 0343

At 500 docs 0 0155

At 1000 docs 0 0083

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 1853

Results from combined n-gram sizes = 4,5,6,7

Queryid (Num) 50

Total number of documents over all queries

Retrieved 333997

Relevant 559

Rel_ret 479

Interpolated Recall - Precision Averages

at 0 00 0 6064

at 0 10 0 5491

at 0 20 0 4158

at 0 30 0 3359

at 0 40 0 2906

at 0 50 0 2607

at 0 60 0 1849

at 0 70 0 1135

at 0 80 0 0799

at 0 90 0 0526

at 1 00 0 0515

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 2369

Precision

At 5 docs 0 2560

At 10 docs 0 2360

At 15 docs 0 2293

At 20 docs 0 1950

At 30 docs 0 1587

At 100 docs 0 0672

At 200 docs 0 0363

At 500 docs 0 0156

At 1000 docs 0 0083

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 2660

Results from combined n-gram sizes = 5,6,7,8

Queryid (Num) 50

Total number of documents over all queries

Retrieved 405629

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 4468

at 0 10 0 3984

at 0 20 0 2871

at 0 30 0 2070

at 0 40 0 1804

at 0 50 0 1546

at 0 60 0 1207

at 0 70 0 0846

at 0 80 0 0648

at 0 90 0 0483

at 1 00 0 0475

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 1677

Precision

At 5 docs 0 1800

At 10 docs 0 1520

At 15 docs 0 1400

At 20 docs 0 1270

At 30 docs 0 1193

At 100 docs 0 0584

At 200 docs 0 0343

At 500 docs 0 0155

At 1000 docs 0 0083

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 1863

Results from combined n-gram sizes = 2,3,4,5,6

Queryid (Num) 50

Total number of documents over all queries

Retrieved 395901

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 5982

at 0 10 0 5424

at 0 20 0 4084

at 0 30 0 3255

at 0 40 0 2844

at 0 50 0 2552

at 0 60 0 1811

at 0 70 0 1137

at 0 80 0 0798

at 0 90 0 0529

at 1 00 0 0519

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 2336

Precision

At 5 docs 0 2520

At 10 docs 0 2320

At 15 docs 0 2200

At 20 docs 0 1930

At 30 docs 0 1580

At 100 docs 0 0672

At 200 docs 0 0365

At 500 docs 0 0156

At 1000 docs 0 0083

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 2624

Results from combined n-gram sizes = 3,4,5,6,7

Queryid (Num) 50

Total number of documents over all queries

Retrieved 405625

Relevant 559

Rel_ret 480

Interpolated Recall - Precision Averages

at 0 00 0 5991

at 0 10 0 5432

at 0 20 0 4088

at 0 30 0 3257

at 0 40 0 2846

at 0 50 0 2549

at 0 60 0 1808

at 0 70 0 1135

at 0 80 0 0798

at 0 90 0 0529

at 1 00 0 0519

Average precision (non-interpolated) for all rel docs(averaged over queries)

0 2334

Precision

At 5 docs 0 2560

At 10 docs 0 2320

At 15 docs 0 2213

At 20 docs 0 1930

At 30 docs 0 1573

At 100 docs 0 0672

At 200 docs 0 0366

At 500 docs 0 0156

At 1000 docs 0 0083

R-Precision (precision after R (= num_rel for a query) docs retrieved)

Exact 0 2624