University of Southampton

Content Based Retrieval and Navigation of Music Using Melodic Pitch Contours

by

Steven George Blackburn

A thesis submitted for the degree of
Doctor of Philosophy

in the
Faculty of Engineering and Applied Science
Department of Electronics and Computer Science

26 September 2000

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE

ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

CONTENT BASED RETRIEVAL AND NAVIGATION OF MUSIC USING MELODIC PITCH CONTOURS

by Steven George Blackburn

The support for audio in existing hypermedia systems is generally not as comprehensive as for text and images, considering audio to be an endpoint medium. Temporal linking has been considered in the Soundviewer for Microcosm. This linking model is beginning to become more mainstream through the development of standards associated with the World Wide Web, such as SMIL.

This thesis investigates the application of content based navigation to music. It considers the viability of using melodic pitch contours as the content representation for retrieval and navigation. It observes the similarity between content based retrieval and navigation and focuses on the fast retrieval of music. The technique of using n-grams for efficient and accurate content based retrieval is explored. The classification of MIDI tracks into musical roles is used to reduce the amount of data that must be indexed, and so increase speed and accuracy. The impact of classification is evaluated against a melodic retrieval engine.

Finally, the system built for retrieval is modified for navigation. A set of tools which support both temporal and content based navigation is built to provide proof of concept.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor, Professor Dave De Roure, for his encouragement, support and direction. It is because of the guidance he gave during my undergraduate project that I considered attempting this doctorate. He has always been available when required through the use of his patented just-in-time supervision. I would also like to thank Dr. Paul Lewis for examining my mini-thesis and providing helpful feedback.

Some key areas of my research would not have been possible without the input of others. Dan Joyce has spent some of his precious time helping advise on (and train) neural networks, for which I am very grateful. Art Pollard of Lextek kindly donated a copy of the Onix text retrieval engine and Pete Elsdon let me capture his knowledge of music. Thank you.

Finally, the nature of the academic environment means that almost everyone in the IAM group has helped me in one way or another.

This thesis was examined by Prof. John ffitch and Dr Paul Lewis on the 13th of December 2000. Thank you for taking the time to read and comment on it.

1 Introduction

1.1 Motivation

A traditional hypermedia system can be thought of as containing a collection of text and images. From an overview document that outlines the subject matter, a user may follow links to alternative, perhaps more detailed, information from the collection. An alternative to starting from an overview document is to search the collection for suitable initial material. There is no widely accepted standard that allows audio to be the subject of this level of user interaction.

In the case of speech radio this might take the form of a user listening to the beginning of a documentary, where the features in the programme are listed, and on hearing a particular topic of interest the user opts to jump immediately to that item; finding an interesting interview abbreviated, the user opts to listen to the entire interview. A synchronized text transcript is available which may also contain pre-authored links and, on hearing an unfamiliar term, the user can ask all available links to be computed dynamically and follow one to a glossary.

Music students could use a similar system as a learning tool. Listening to set pieces of music is often part of their course work. The music can be distributed over a network from a digital music library, such as the VARIATIONS project at Indiana University [Dunn99]. On hearing an unfamiliar concept, the student could find information by following links that have been previously authored by the music lecturer. This can be very powerful if the links are based on the content of the music, so that the link is available for every occurrence of a theme. Retrieval of music using pitch-based queries could be used to locate similar music. This could also be used as a research tool or as an aid to composition; it allows the composer to ask ‘how have others developed this theme’.

Hypermedia systems generally consider audio as an endpoint medium. The hyper-text markup language (HTML), as used on the World Wide Web, is an example of such a system. Text and images, through the use of image maps, may link to audio but there is currently no reciprocal linking mechanism for audio. It is not possible to be listening to a piece of music and to follow a link from an interesting segment to some related information.

Temporal linking has been considered in the Soundviewer [Goose95] for Microcosm [Fountain90]. This linking model is beginning to become more mainstream through the development of standards associated with the World Wide Web, such as SMIL [SMIL]. Here, links may be created that are based on playback positions within a media document. Literature regarding links based on features extracted from audio documents is almost non-existent, with MAVIS [Lewis96a][Lewis96b] being an exception.

Current literature on audio generally consists of systems which classify and catalogue short digital audio samples (Foote [Foote97], Wold *et al* [Wold96]), with the aim of helping retrieval. Classification and retrieval of music has also been examined (Pikrakis *et al* [Pikrakis96], Ghias *et al* [Ghias95], McNab *et al* [McNab96]). The only systems that consider the navigation of audio, and where links may be based on features extracted from audio, have concentrated on extracting and making use of consequential information that, while identifying content, does not describe the content. As such, these systems provide limited support for content based navigation.

1.2 The problem

One can consider retrieval as returning the names of documents which *contain* the query content. Navigation can be considered as returning the names of files which are *associated with* to the query content. Both applications can use the same content indexing principles, but to reference different types of data.

For an interactive system, like Content Based Navigation (CBN), lookup must be fast as queries are made in real-time (as opposed to off-line or batch). Waiting more than a couple of seconds for the query to complete will be seen by the user as an inconvenience.

Retrieval may involve large collections of music, perhaps of the order of 100,000 pieces in the case of an Internet-based search engine. Navigation is likely to have more moderate requirements, as only segments of music will be indexed.

Approximate queries must be supported. The quality of the music files in the system can not always be guaranteed: they may contain errors or there may be several arrangements of the same piece. Errors in tune recall on the behalf of the user (in the case of a user-entered query) or in the entry of the query must also be accounted for.

Many content representations for music have been considered by researchers in the field of content based retrieval. This thesis examines the effect of using melodic pitch contours as the content representation. A melodic pitch contour describes a series of relative pitch transitions. A note in a piece of music is compared with a previous note and then classified in one of three ways: either the pitch went up (U); the pitch went down (D); or it is a repetition (R). Thus, the piece can be converted into a string with a three-letter alphabet (U, D, R).

Musical pitch contours are easily calculated from pitch information, such as that contained in files conforming to the Musical Instrument Digital Interface (MIDI) standard. These files specify, among other things, when a note is to start, what pitch and volume it is to be played at and when to stop each note. Both pitch contours and MIDI files assume that all music consists of notes which, while not universally applicable, covers the vast majority of modern western music. This is assumed for the majority of this thesis. Techniques that are suitable for other forms of music are noted where appropriate.

1.3 Contribution

This work shows that melodic pitch contours are sufficiently powerful for retrieval and navigation of music. A specialised data format is designed and approximate string matching techniques are employed. This format is shown to be superior to other standard database formats with respect to the speed of retrieval.

Secondary contours are introduced as a new and complementary representation that does not require the development of additional indexing techniques. They are shown to improve matching while adding little overhead to the search process. Using a secondary contour in parallel with a primary one is shown to be more expressive than using a representation that has five pitch classes.

Standard classification algorithms are applied to classify the role of each track in a MIDI file. This is then used to enhance the music database by only indexing tracks which are likely to be the subject of a query. The effectiveness of this approach is evaluated.

The architecture for a system that demonstrates temporal navigation of temporal media is designed and implemented. This architecture is then extended to support links based on the content of music-based media (demonstrated on MIDI files).

1.4 Thesis structure

This thesis presents techniques which enable the use of multimedia, specifically audio and music, as the basis of a document. A review of the related work is given in Chapter 2, followed in Chapter 3 by an explanation of the author’s work on content based retrieval of music. Classification of music is considered in Chapter 4, along with the effect of applying it to a content based system. The application of pitch contours to content based navigation is covered in Chapter 5. Finally, Chapter 6 presents the conclusions and possible future work.

2 Related work

This chapter discusses research relating to the topic of this thesis. It starts with an explanation of the term ‘navigation’ and what it means to navigate audio. Music representations are then considered before examining content based retrieval and navigation. A discussion of open hypermedia and other issues relating to audio concludes this chapter.

2.1 Navigation

Anyone who has used the World Wide Web is familiar with navigating around a body of documents. Links are created either between or within documents by embedding information at relevant points in the source document, indicating material related to that point. The Hyper Text Mark-up Language (HTML) [HTML4] on the world wide web is the, now classic, example of this model. An information space is navigated by activating a link in a document, which then takes the user to the related document.

This approach to organising an information space was first proposed by Vanevar Bush in 1945 as a way of managing information overload [Bush45]. The concept was first applied to computer documents over 30 years later in Ted Nelson’s Xanadu which considered the links between documents to be of upmost importance [Nelson81]. This too was not a working system but simply a mock-up of what it could look like.

The navigation systems described here differ in where the link information is stored. One approach is to embed the links in the source document, as is traditionally the case with HTML, although XML is changing this through XLINK and XHTML. The other approach is to keep the links separate from the documents.

In all cases, the concept of a link is the same although different terminology is often used. A link contains at least one source endpoint and at least one destination endpoint. An endpoint specifies a location in a document, or perhaps a whole document. In this way, two endpoints are associated with each other so that, via an appropriate user interface, the source endpoint may be clicked on and the destination endpoint displayed[1]. Some systems support links with attributes, which allows for more powerful interaction with a document, but the concept of associating two or more entities is common to all.

There are a couple of problems with the use of embedded links. The first being that, while it is possible to link *to* any media, the source document is usually text (although image maps are quite common). It is currently not possible to be listening to a piece of audio and link to a web document that is related to the clip currently being played. This issue is gradually being addressed through standards associated with the web, such as the Synchronised Multimedia Integration Language (SMIL) [SMIL] and associated editors [Bulterman98].

The second problem is that all instances of conceptual links must be manually implemented. For example, if one wished to link every occurrence of the phrase "University of Southampton" to the home page of the University, links must be created for each occurrence of the phrase. If the location of the University page changes, all instances of the links must be found and modified.

Searching can be considered as another mode of navigation. The user finds a phrase of interest and searches a body of documents for other occurrences of that phrase. This can work well in the absence of other linking mechanisms, as it is reasonable to assume that documents containing similar phrases may concern related subject matter. The technique was used several times to locate some of the information contained in this chapter. These links are said to be generated automatically, as they do not require human interaction to author them. This approach is not foolproof as a search for the phrase "baby kangaroos" will, along with relevant documents, find documents that contain the phrase "this document

[1]It should be noted that this is just one method of interacting with a link, but it is the most commonly used and is the model employed by the World Wide Web. In HTML, the source endpoint is implied by the location of the link in the document. Also, only one endpoint may be specified (i.e. it only supports binary links).

is not about baby kangaroos”.

The process of searching usually uses an index of the documents that is periodically generated. This speeds up the process and can be used to centralise resource, as is the case with Internet search engines. The ‘search as navigation’ paradigm can be refined, so that the document index is not generated automatically and, rather than identifying documents that contain a phrase, returns a list of documents that are related to that phrase. This model of hypermedia is referred to as Open Hypermedia and is discussed in Section 2.10.

2.2 Navigation of audio

The ability to link to specified parts of audio files, and to link out from audio files (for example to the current location in a text transcript of a speech), is not very common but nonetheless desirable, as the scenarios given in the introduction show. The simplest form of this is temporal navigation.

In temporal navigation, links are based on the start time and length forming a selection in a temporal media document, such as audio or video. These links are usually stored separately from the media as not all media formats support additional embedded information. Upon finding an interesting segment in the media, as indicated by the user, a simple database query is performed to identify any links appropriate to the timing information of that media segment.

The Microcosm Soundviewer was an audio tool which supported hypermedia links into and out of stored audio via this mechanism [Goose95]. Using the Microcosm open hypermedia system [Davis92][Fountain90], links could be authored based on the position of a selection in a document. All the links from a particular audio document were found before playing began. These links were displayed as coloured ribbons in an area above the media time line, see Figure 2.1. These ribbons identified the times over which a link was active. The Soundviewer also had an option where a link could be automatically activated upon the link becoming relevant. The two-dimensional display of the links takes up valuable space on the screen, and can easily become cumbersome where a document has many links. It has the advantage of making the concept of temporal hypermedia easy to understand.

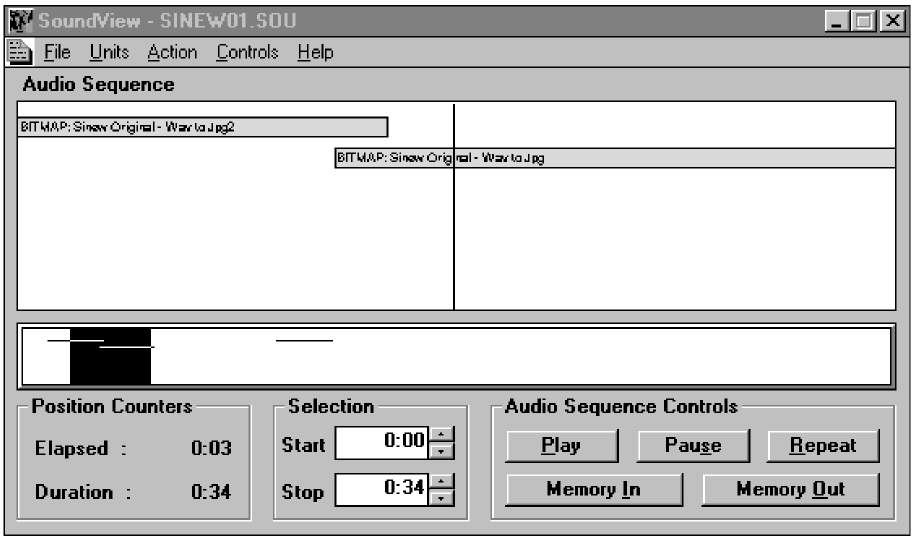


Figure 2.1 - The Microcosm Soundviewer

An audio only environment was developed by Sawhney and Murphy, called Espace 2 [Sawhney96]. It was a prototype system for navigating hyper-linked audio documents. Continuous audio streams were used to represent the availability of links to other documents. The example content used recordings of conversations and discussions. It is likely that the links were time based, due to the concentration of the project on the user interface rather than content extraction.

Buchanan and Zellweger examined ways of specifying temporal behaviour in hypermedia documents [Buchanan92]. The Firefly document system provided support for synchronisation of media clips in a presentation similar to that provided by SMIL (see below), i.e. allowing clips to occur in sequence or in parallel. Again, links between clips were time based, although the exact timing was only resolved when the document was displayed, as opposed to being an authoring process. Authors needed only to specify event relationships (events could occur either in sequence or in parallel).

Using only timing information for linking works well with any temporal media, such as video, as the content is not considered. However, it suffers from one of the problems of embedded links: each link must be manually authored and maintained. To overcome this, links need to be based on the content of a selection in the document.

For links based on content, the source endpoint is specified by some example content, instead of a position in a file. Resolving a link requires the content of the source selection to be extracted and compared with the source endpoints in a link database to find relevant links. This is the intended use of the Open Hypermedia model, as discussed in Section 2.10.

2.3 The Synchronised Multimedia Integration Language

The Synchronised Multimedia Integration Language (SMIL, pronounced ‘smile’) is an XML-based initiative that aims to provide web-based interactive multimedia applications [SMIL]. The display is split into regions through the use of a layout specification that is given in the head of a SMIL document. Multimedia files or streams can be assigned to these layout elements. The flow and synchronisation of the presentation is controlled by specifying which elements must be displayed in parallel and which must be displayed in series.

Version 1.0 of the SMIL standard specifies two methods of embedding uni-directional links within a presentation. The first is the “a” element which is intended to be the same as the “a” element in HTML documents. It allows a link to be formed from a whole media object. It is not possible to describe a link that is active for a specific sub-part of the media object. The “anchor” element addresses this shortcoming by providing support for image maps and the temporal equivalent.

Multiple anchors can be associated with each media object. Each will specify some location within that object. This will be coordinates for images, begin and end times for audio and may be both for video. When used as the source of a link, the anchor element will contain a destination Universal Resource Indicator (URI).

The destination of links uses the Universal Resource Locator (URL) format that was first used in HTML and has now itself become a standard. Complete documents or presentations may be referenced in the usual manner. Parts of documents or presentations may be addressed through the use of id attributes and the “#” connector. Each media object, anchor and “a” element of a SMIL presentation may be given an id, e.g. id = “conclusion”. Individual elements can be addressed by appending a “#” followed by an id to the URL, e.g. http://www.w3c.org/Presentation#conclusion.

These elements and attributes allow SMIL to be used to specify one-to-one links that must be embedded in the source document. Any element, or part thereof, that is the destination of a link must be given an identifier. While this is the most comprehensive mainstream support for linking audio currently available, it does have some drawbacks. Perhaps the biggest is that it is impossible for an author to foresee all the parts of a presentation that may be the destination of a link and so give it an identifier. This is an especially important point when the person wishing to link to the presentation can not directly modify the file. The use of embedded links in SMIL results in the same drawbacks as that shared by HTML.

Support for SMIL is growing, with the availability of both editors and players increasing. The official web site for SMIL lists eight players (eg: RealMedia's RealPlayer G2) and ten authoring tools (including GRiNS [Bulterman98]) that support SMIL in some capacity. The biggest indication that SMIL may receive wide acceptance is the introduction of basic support for the standard in the latest version (v5.5) of Microsoft's Internet Explorer. A product with a large market share is generally required before people will see any advantage in using a new standard. There is usually little sense developing a presentation that can only be viewed by a handful of people.

The latest working draft of the standard, codenamed SMIL Boston [SMIL20], has taken the same constructs but modified the XML tag names to be more parser friendly. There is an XML parser for nearly all programming languages and platforms, but these are not all of a high standard. The most noticeable change to the SMIL specification is the removal of hyphens from attribute names. Mixed case tag names have been used instead, so 'clip-begin' becomes 'clipBegin'. The move to incorporate more of the XML related standards has resulted in the adoption of XPointer as a way of specifying an element that is the destination of a link without requiring the subject element to have an identifier.

A new synchronisation type has been introduced which is similar to the parallel mode but only allows one element to be active at any given time. This is intended to cater for scenarios such as adding audio descriptions to a video stream to help the visual impaired. The video stream would pause while the description is playing and then resume.

2.4 HyTime

The Hypermedia/Time-based Structured Language (HyTime [HyTime]), can be thought of as the precursor to SMIL in some respects. HyTime extends the SGML standard to define ways of representing the information elements required by a hypermedia systems. It was the first international standard to consider the representation of hypermedia objects (as opposed to hypertext objects).

A HyTime document consists of a meta-DTD (Document Type Definition) that defines the set of rules used by the document object, which is followed by the document objects themselves. This makes HyTime flexible, as the document format is essentially ‘programmable’, but makes it difficult to use, as any application that supports HyTime must incorporate a full SGML parser to be able to make sense of the document. This also makes it difficult to author HyTime documents, as the document rules must be specified before proceeding with writing the document body. This may be an acceptable cost in a large hypermedia system where the HyTime document is a large one and so the relative time taken to define the rules is negligible. This gives an insight into what was the likely target application of the HyTime standard, as the storage layer of a large hypermedia application.

All the above factors make it an unsuitable language for the World Wide Web. According to Kimber [Kimber96], “for many applications, it will make sense to transform SGML- and HyTime-based information into some optimized or proprietary form for final delivery.”He gives an example of a HyTime document being rendered to HTML for use on the Web.

It is clear that HyTime is a complex standard which, while flexible and general, suffers for this. There are very few applications which implement the standard, although there are more SGML-based editors which help an author of a HyTime document. It is also clear that SMIL is easier to understand and simpler to write. The ordering of objects in HyTime requires defining an axis (in this case ‘time’) before objects, or groups of objects, can be placed on that axis.

SMIL is simpler than HyTime but is not as flexible or generic. These qualities make it a standard that is suitable for the World Wide Web. It standardises a straightforward approach that can be implemented relatively easily.

2.5 Content representations

Previous sections have discussed several uses of media content in retrieval and navigation. How this content is represented is important, as it affects the choice of algorithms that may be used for resolving content based links. For example, text is a suitable content representation for a recorded speech and many text retrieval algorithms already exist. Some appropriate representations for music, and their suitability for comparing / matching music, are discussed here. Most of these can be obtained by performing feature extraction on another, usually lower level, representation. As such, feature extraction can be thought of as representation conversion, taking a low level representation and identifying higher level features.

Wiggins *et al* [Wiggins93] described a framework for evaluating music representation systems. They considered a representation to be useful for one, or more, of three purposes: recording, analysis, or composition. They examined the expressive completeness (the range of musical data which may be represented) and the structural generality (the range of musical structure) of each of the representations. Precise values for completeness on generality were never assigned for any of the models they looked at, preferring to use these concepts to order the representations in two-dimensional space.

The representation used for content based retrieval and navigation systems may depend on where the query comes from. For example, a representation which allows quick, exact, matching might be used when linking from selections in MIDI files, as a precise query can be expected.

The range of currently available content representations is diverse. Some represent essentially the same data, perhaps in a format suitable for a specific project. As such, an exhaustive survey covering all representations of music and audio that are relevant to this thesis is both impractical and of little use. The following sections give an overview of the properties of some of the representations used for music in its different forms: from composition to a physical sound wave.

It should be noted that some of the representations discussed here (e.g. MIDI) assume that music consists of melodic note events. As explained in the first chapter, this assumption must hold true for melodic pitch contours, the subject of this thesis, to be applicable.

2.5.1 The musical score

The musical score is a representation used by a composer to instruct a musician on how to perform a particular piece of music. As such it is the definitive document which conveys all the notational nuances which are lost as soon as the piece is performed. It is also likely to contain some structural information which is also lost in performance, such as repeating bars. There are many existing file formats which may be used to store musical scores, such as the Standard Music Description Language (SMDL) [SMDL] which is an SGML-based format. The expressive completeness and structural generality are high.

2.5.2 Performance data

The Musical Instrument Digital Interface (MIDI) [MIDI] is a standard which allows digital instruments to transmit performance data between each other. This performance data may be stored in MIDI files (see section 2.13), so recording the musician’s performance. The MIDI file format is a linear, time stamped, sequence of events (a detailed explanation is in Section 2.13).

Due to the performance being a musician’s interpretation of a musical score, one could argue that more information is contained in the MIDI stream. However, a MIDI stream can also be derived from the score and, due to the linear nature of the recording, a lot of structure and markup is lost. It is more accurate to say that the MIDI representation may contain different information to the score.

2.5.2.1 Notational differences

Most performance oriented standards are concerned with exact pitch, rather than symbols on a stave, being communicated and stored. This results in $C^{\#}$ and D^{b} being stored as the same pitch (which they usually are). This difference is not important to the listener but it is to musicologists. Although similarity matching of music concentrates on pitch, comparison of the notation might allow better use of the existing data. This requires an algorithm for determining the correct pitch notation. This issue has been investigated by Blombach [Blombach95], who claims to have developed an algorithm that achieves this.

Hewlett also explains the usefulness of retaining the ‘spelling’ of pitches and notes that 40 enharmonic spellings are required per octave [Hewlett92]. He developed a base-40 representation which was used as part of a complex binary format that allowed a small set of classical music to be searched on a machine with limited memory. The base-40 form is the most complete one found for notes and is aimed at people who wish to analyse pieces of music.

2.5.3 Melodic pitch contours

Music is recognisable irrespective of the key it is played in, a property which shows that matching on the exact sequence of notes is not desirable. Exact intervals can be used for matching, identifying the interval (in semitones) between the current note and the last, as this is independent of key. Dowling suggested that the pitch direction, simply noting whether the pitch went up, down or was repeated, is useful in some situations [Dowling78]. This representation is referred to as a melodic pitch contour. Lindsay discussed how pitch contours are not a new idea, indeed they were used as a form of musical score for chants [Lindsay94].

A melodic pitch contour describes a series of relative pitch transitions, an abstraction of a sequence of notes. A note in a piece of music is compared with the previous note and then classified in one of three ways: either the pitch went up (U); the pitch went down (D); or it is a repetition (R). Thus, the piece can be converted into a string with a three-letter alphabet (U, D, R). For example, the introductory theme to Beethoven's 5th Symphony would be converted into the sequence: - R R D U R R D. Notice that there is one less symbol than notes as only the transitions between notes are recorded. An example of a musical score and its associated pitch contour is shown in Figure 2.2.

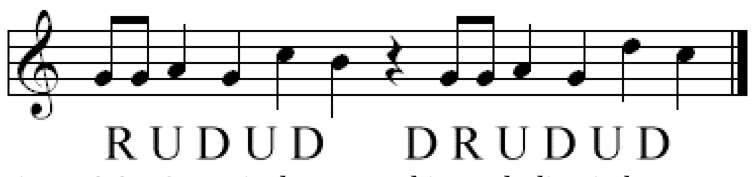


Figure 2.2 - A musical score and its melodic pitch contour for “Happy Birthday”

Melodic pitch contours discard a lot of information which might otherwise reduce the search space, such as rhythm information and the size of intervals between notes. This results in the representation being an under-specification, effectively having a certain amount of approximate matching built in. If the notes in two pieces of music rise and fall in the same order then they are considered as matching, regardless of the amount of change in pitch.

The melodic pitch contour described above is not likely to discriminate enough in some circumstances (e.g. comparing short clips of highly similar music). The other extreme is to use the exact size and direction of an interval between notes. The compromise is to use a number of interval classes, so that the UDR representation is considered to have three interval classes. The number of classes must be odd if one is used for repeated notes while the remaining classes are split equally to represent both positive and negative intervals[2]. This thesis refers to the three-class representation as being a gross melodic pitch contour, due to its very general nature. Downie has evaluated the effect that the number of pitch classes has on recall and precision in his MusiFind system (more below).

2.5.4 Rhythm contours

The rhythm of a piece of music can be generalised in a similar way to musical pitch contours. The length of a note in a piece of music is compared with the length of the previous note and then classified in one of three ways: either the note was longer (L); the note was shorter (S); or it is the same length (R). Thus, the piece can be converted into a string with a three-letter alphabet (L, S, R). When determining the rhythm class of a note, an allowance must be made for small differences in the lengths of ‘repeated’ notes. The obvious example of where this is needed is for music that is entered by hand on a piano keyboard; reproducing notes of exactly the same length is beyond the capabilities of a person. A common practice is to base the threshold on the length of the previous note. If the length of the current note is half that of the previous one then class the current length as shorter. If it is twice that of the previous one then it is classed as longer. Remaining

[2]Technically, there need only be more than one pitch class if more non-conventional assignment is used, such as ‘up’ and ‘not up’. However, no research has been published which either suggests, uses or evaluates this approach.

notes are said to have the same length.

Unlike melodic pitch contours, this representation is not grounded in psychology theory but is intended to make finding similar pieces easier. It should be noted that there is no way to keep rest information. It should also be noted that it is common to use the note onset interval as the length, i.e. the distance between one note starting and the next note starting. This is often to overcome a limitation of MIDI where a note may sound for a while after the ‘note off’ signal is received due to the sustain of the current instrument. The problem is also present on a real piano, where the note may still sound even though the key has been released. One can not be sure what the user has heard as a result of this ambiguity, so the start of the next note is used as a concrete event.

Arbee Chen [Chen98a] has examined the use of rhythm as a means of retrieving songs. This used the note durations as given by a musical score and included a representation of rests. Analysis showed that five mubols, the symbol for one note, were enough to uniquely identify one song out of 102 folk songs.

2.5.5 The Fourier transform

The Fourier transform relies on the principle that any waveform can be described as the sum of component sine waves, each at a particular frequency, amplitude and phase. The Fast Fourier Transform (FFT) is an efficient algorithm which approximates the Fourier transform for sampled signals by using a divide and conquer technique. Given a digital audio file, a table which identifies the amplitude of each component frequency, at any moment in time, may be calculated. An inverse of the transform also exists, so that digital audio may be reconstructed from a frequency spectra.

There is an uncertainty principle that affects the precision of the FFT algorithm. There is a trade off between the accuracy of the timing and the accuracy of the frequency table. The signal must be analysed in n-sample sized chunks (called windows). Small windows allow the timing to be accurate, but the small number of samples limits the ability to accurately identify the frequency. The inverse is true for large windows.

The FFT is directly related to the digital audio representation, functionally carrying the same information, so it can be used to express almost anything. However, like digital audio, it lacks explicit musical structure.

The Fast Fourier Transform calculates the Discrete Fourier Transform (DFT). Another commonly used transform is the Discrete Cosine Transform (DCT). It is from the same family as the DFT, essentially being a simplified version of the DFT.

2.5.6 Digital audio

The rawest form of representation is obtained by digitising an audio signal. The amplitude of a signal is converted, from an analogue form to a digital one, several thousand times per second. For example, professional audio recordings currently take 48,000 or 96,000 samples per second. The process can be inverted to reproduce a signal very close to the original. Once in the digital domain, the audio data may be subjected to lossless or lossy compression.

A common storage format for digital audio is the wave file. This is a Resource Interchange File Format (RIFF) based format that may contain meta-data along with the digital audio, although home users tend not to use this feature. It is possible to extend the format to contain alternative representations discussed above.

A lossy compressed form of audio that is rapidly gaining popularity is the audio layer (layer 3) of the MPEG-1 specification (see below), commonly referred to as MP3. This uses psycho-acoustic masking to reduce the amount of information about the signal that needs to be stored. It was originally developed as part of the audio/video compression standard used by VideoCDs. It has gained a lot of support due to the high compression ratio (12:1) that still retains CD quality sound (as perceived by a panel of listeners). The reason for the recent increase in the use of MP3 files is probably due to the ability to transmit audio over the Internet in near-CD quality sound, assuming a reasonably high quality home connection (e.g. a cable modem or ISDN connection).

2.5.7 Meta-data and associated standards

Much of the information which content based systems have to work hard to extract was probably initially available when the document was authored. Having some way of keeping this information would improve the quality of content based applications. The answer is to keep meta-data, that contains descriptions of the content, with the distributed media. Of course, this can only apply to new media; historical archives will still require the content to be recognised before meta-data can be stored.

Meta-data is most generally described as data about data. What this actually means will depend on the data type and how it is being used. This may be bibliographic information, such as the title and year of publication of a song. The term also refers to alternative representations of the material that may be used for analysis or search.

2.5.7.1 Meta-data in digital audio files

There are several extensions to the MP3 file format that provide varying amounts of meta-data. The simplest, and most common, is the ID3v1 tag which may be found at the end of many MP3 files. It contains basic track information: title; artist; album; year and a comment. These fields may contain up to 30 characters, the exception being year (four characters).

The overly specified ID3v2 tag may be found at the beginning of the file and is designed to be flexible and overcome the limitations of the original ID3 tag. There is no limit on the size of fields and new fields may be added simply. Although it allows much wanted fields, such as remix artist and album artist, and removes the constraining field size limit, it is unlikely to be widely accepted. This is due to the format being very complex and the tag being stored at the beginning of the file, making modifications to the tag hard-disk intensive.

The biggest problem with the first version of the tag was the limited field size. There is a new meta-data format emerging for MP3 which is as simple to manipulate as the ID3v1 tag but allows 256 characters per field. It also includes support for embedding lyrics in the file and so is referred to as the Lyric3.2 tag. There is a standard method for time stamping the lyrics which, while designed for karaoke files, provides a rich set of meta-data for content based retrieval and navigation systems.

2.5.7.2 Associated standards

As part of the International Organisation for Standardisation (ISO), the Moving Pictures Expert Group (MPEG) has developed a family of multimedia standards. These are:

- Storage and retrieval (MPEG-1)
- Digital television (MPEG-2)
- Multimedia production, distribution and content access (MPEG-4)
- The multimedia content description interface (MPEG-7)
- The multimedia framework (MPEG-21)

The first of the family, MPEG-1, was designed to allow storage and retrieval of moving images and accompanying audio. The aim was to provide a data stream that required a bandwidth that was no more than the data access rate of a CD-ROM drive contemporary with the year of publication of the standard. The MPEG-2 standard strives for a higher resolution, and therefore a higher bit rate, that is comparable with that used in digital video studios. MPEG-2 can be thought of as the next generation of the MPEG-1 standard. While this is somewhat of a simplification, MPEG-2 remains backward-compatible with MPEG-1. Both standards address the coding and transfer of low-level data for the purpose of storage and playback. As such, they are not directly related to this thesis, other than that they may be used to encode the temporal media discussed herein.

The MPEG-4 standard moves away from the encoding of the media to consider the process of multimedia production. Nack and Lindsay [Nack99] sum up the aims of MPEG-4:

> "MPEG-4 aims to provide a set of technologies to satisfy the need of authors, service providers, and end users, by avoiding the emergence of a multitude of proprietary, incompatible formats and players."

The part of the standard that is most interesting in the context of this thesis is the ability to attach textual meta-data to streams. Unfortunately, the format of the textual information is not defined, which limits the long-term usefulness of this feature.

MPEG-4 also standardises several different tools for decoding both natural and synthetic sounds. One of these tools being Structured Audio which allows for audio to be represented such that it can be synthesised by an MPEG-4 compatible player. The data stream describes a *score* and an *orchestra*. The score identifies the notes to be played, while the orchestra describes the timbre of each note. Again, this is a gross simplification of the standard but summarises the points of interest here. For a more detailed discussion of MPEG-4, the reader is directed to Scheirer [Scheirer99].

The issue of describing multimedia material is being addressed by MPEG-7, which is currently under development. The aim is to make multimedia documents a searchable as text by fully describing any multimedia content. It hopes to define a framework for defining structures for meta-data, similar to the way in which the Standard Generalised Mark-up Language (SGML) can be used to define HTML. In MPEG-7, an attribute of some media (a single piece of data) is called a descriptor. A set of attributes is called a description scheme. The standard will detail an XML-based Description Definition Language (DDL) which may be used to define a description scheme. The DDL might also allow definitions of descriptors, this is currently under discussion. Several descriptors and description schemes will be included, something that will be required to ensure some compatibility between MPEG-7 compliant systems.

Work began in October 1999 to define a framework for multimedia, MPEG-21. According to the latest working paper, the aim of the initiative is:

> "To reach a common understanding about a multimedia framework to support the delivery of electronic content with the purpose of identifying where new standards falling under the competence of SC29[3] are required."

The goal of MPEG-21 is to define a set of requirements for components of a "multimedia delivery service" to interoperate. The first draft will look at the problem from the perspective of the consumer. One of the twelve key issues that were raised during a brainstorming session was "searching, filtering locating, retrieving and storing content".

2.6 Content based retrieval of digital audio

Content based retrieval is important to content based navigation because the issues of content representation, and techniques for indexing and performing approximate matching

[3]SC29 stands for subcommittee 29, which forms part of the Joint ISO/IEC Technical Committee on Information Technology. The Moving Picture Coding Experts Group (MPEG) is officially working group 11 of SC29.

on that representation, have to be considered. Indeed, the difference between retrieval and navigation is quite subtle: items *containing* the query are located in content based retrieval, while items *relating* to the query are returned in navigation. As such, before examining content based navigation, it is useful to investigate content based retrieval systems. Retrieval has the additional use as a means of locating an initial media document, required before the document space can be navigated.

2.6.1 Retrieval of digital audio samples

The term “digital audio sample” is used here to refer to short, descriptive, sounds such as a bird chirping, a bell ringing or a car starting. Foote [Foote97] and Wold *et al* [Wold96] described systems which could retrieve such samples. The general approach was to calculate the Fourier transform of a sample then extract some salient features from it. These features could then be approximately matched against a database of pre-processed samples.

The Fourier transform works well for this application as the samples are short (making comprehensive indexing possible) and distinctive. The matches returned are likely to be very specific, i.e. returning the sound of a specific car (e.g. a Ford Escort) starting rather than the sound of any car starting. This is because the frequency spectra resulting from the transform can be very detailed and precise. Approximate matching can be used to ignore some differences but will match irrelevant samples if the matching is too approximate.

2.6.2 Retrieval of digital audio media

The distinctive nature of frequency spectra has been used by David Gibson to develop his ‘name that clip’ software [Gibson99]. It is used to index collections of digital audio, such as CD-Audio. The Fourier transform of the audio is taken five times a second and added to a database. Given a query, the same frequency analysis is performed and the database searched for the nearest match. The problem of searching can be considered one of a 160-dimensional nearest-neighbour search. The current implementation is in Java and is too slow for interactive use (20 minutes). The accepted limitation of the software is that the query must be a recording of the original audio. A recording from a radio transmission or a microphone near a speaker is acceptable, but a hummed query or a piano rendition will not suffice.

2.6.3 Applying image matching techniques to audio

Some concepts of image matching have been applied to audio classification / retrieval. This is useful for analysing digital input, such as from a CD, as there is currently no general purpose (and reliable) method of converting multi-timbral, polyphonic, music into a musical score. This prevents the melodic retrieval techniques discussed in following sections from being applicable, although we would still like to be able to index and search the music in a similar manner. Martin [Martin96] details the ongoing research into polyphonic pitch tracking.

Image recognition allows a user to sketch a picture which is then analysed and compared with images in a database. The images are indexed by identifying features in them, such as lines and circles and their relationship with each other. If a picture could be created which described a piece of music then features of that picture could be extracted and used to index the music.

Creating an image of sound can be achieved by use of the Fast Fourier Transform. The frequency vs. amplitude table can be plotted on a two-dimensional graph, time against frequency, where the amplitude is represented by using a colour scale. An example of the resulting picture is shown in Figure 2.3. By applying image matching techniques, this representation can then be used for indexing. This is a useful technique as it can index any form of digital audio although it tends to be used for classifying music. Pikrakis *et al* [Pikrakis96] reported on one such classification project that tries to identify forms of Greek traditional music. Fushikida *et al* [Fushikida98] used an FFT plot in conjunction with video to improve content based video scene retrieval.

A similar technique has been adopted Attrasoft[4], the authors of the shareware program MidiFinder. The program is trained using one or more MIDI segments for which similar segments are desired. A directory of MIDI files is scanned for files containing matching segments. According to the user guide, the software is related to the ImageFinder program that is produced by the same company. MIDI files can be considered to contain several one-dimensional data arrays. They are converted to two-dimensional images by using “1

[4]Attrasoft could be found on the Internet at http://attrasoft.com at the time of writing

to 16 pixels per quarter note". A quarter note is the unit of time represented in a MIDI file by a crochet. MidiFinder forms part of the SoundFinder family of products.

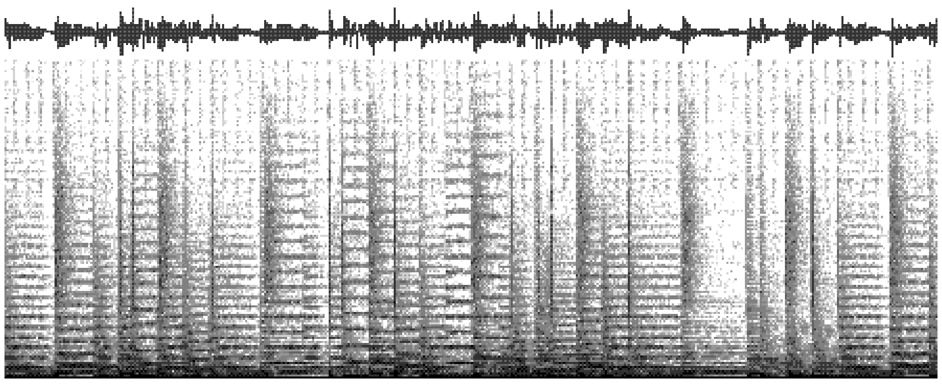


Figure 2.3 - The output from a Fast Fourier Transform plotted on a two-dimensional graph

2.7 Melodic retrieval systems

The increase in interest in the field of music retrieval systems is reflected by an increase in the amount of research and software concerning melodic retrieval. Older systems were designed to be used by musicologists for analysing pieces of work, whilst more recent work is more user-oriented and has a commercial edge. The commercial aspect is evident with several systems having World Wide Web interfaces.

Query by humming is possibly the definitive application of content based retrieval of music. In such a system acoustic input, typically sung or hummed by the user, is transcribed and songs containing the input are retrieved from a collection of music. Results are ranked according to how closely the songs match the input. Allowance must be made for inaccurate singing or errors in the tune recall of the user.

A number of melodic retrieval systems are discussed here which show some of the approaches taken by other researchers in the field and the resulting performance (when given). The systems vary in their goals, so concentrate on different aspects of a retrieval system. The research relating to the MusiFind system is referred to in later chapters, as it has a thorough statistical analysis of melodic contours.

2.7.1 Catfind

Catfind is an Internet search engine for music, based at the University of Hong Kong and written by Yip Chi Lap [Yip99]. The first version could search by gross melodic contour or a solfa description (e.g. do-re-me). Contours used the characters '/', '-' and '\', so the Happy Birthday example above would be represented as '-/\/\\-/\/\'. The current version does not support melodic contour entry but still allows this as a search mechanism (i.e. only pitch direction is taken account of when performing the search).

2.7.2 Echo

Echo is a 'search by humming' system that was developed by Tomonari Sonada *et al* at Waseda University, Tokyo [Sonoda98]. Pitch tracking is performed by a client program which submits a query to a server on the Internet to retrieve songs matching the query.

The Echo system uses two techniques to improve the quality of results. The first is to optimise the boundaries for pitch classes, dynamic thresholds. Sonoda argues that it is beneficial to have a more flexible concept of 'up' and 'down'. Instead of rigidly fixing 'up' to be any pitch that is a semitone or greater than the preceding one, the intervals of the indexed collection are analysed to determine thresholds that discriminate the most. The same is done for rhythm contours.

The second method of improving the quality of matching is to use progressively more pitch classes until few (or just one) files match. This is referred to in the paper as coarse-to-fine matching. A search is first performed with three pitch classes (a gross melodic contour). If too many files match then a search with nine pitch classes is performed, and finally with twenty-seven classes.

Table 1 - The effect of using dynamic thresholds on recall

Threshold	Pitch only	Rhythm only	Both
Static	47.3%	13.4%	90.2%
Dynamic	49.1%	35.1%	97.3%

A user trial was performed with a total of 112 queries from 12 subjects on a database of

200 songs. Table 1 shows an increase in recall when using dynamic thresholds for the gross melodic contour and rhythm contour (source: Sonoda [Sonoda98]). The technique has the most impact on rhythm contours. A decrease in accuracy was shown when using coarse-to-fine matching and searching on either pitch or rhythm. An increase of less than one percent was shown when matching using both pitch and rhythm. The coarse-to-fine matching proved beneficial when errors were intentionally introduced into the queries. Increases in accuracy were then made apparent, although the overall accuracy was almost 4% less than with no errors.

Combining the two methods and searching using both pitch and rhythm resulted in 100% of queries ranking the subject files in the top three suggestions.

The system was then expanded to contain 10,000 songs and methods of indexing large music databases investigated [Sonoda2000]. They claim an average execution time of 0.78 seconds per query and an accuracy of 92%.

2.7.3 Humdrum

Humdrum is a set of tools used in the research of music, developed by David Huron at Ohio State University [Huron97]. These tools operate on data conforming to the Humdrum Syntax, best described as a generic representation framework. It allows many forms of audio to be stored so as not to limit the scope of the tools, although the most well-known format is “**kern”. The kern representation is designed to convey functional information about a piece of music. It is not intended to describe a score for printing. The syntax also specifies a number of reference records, which may define bibliographic information pertaining to a piece.

The system has been used by musicologists to answer a wide variety of questions about music. Two of the examples given in the user guide are: “In Urdu folk songs, how common is the so-called "melodic arch" – where phrases tend to ascend and then descend in pitch?” and “What are the most common fret-board patterns in guitar riffs by Jimi Hendrix?”. The tool of interest to this thesis is ‘patt’ which will “locate and output user-defined patterns in a Humdrum input”.

2.7.4 Lemström and Laine

Lemström and Laine ignored the pitch tracking to focus on the retrieval process [Lemstrom98]. They presented a representation for musical data which they called the inner representation. A two-dimensional relative code, derived from the inner representation, was used for the matching as it was independent of the original key. An efficient indexing structure, the suffix-trie, was used in the matching phase.

The music is indexed using relative pitch changes and duration between the starts of notes (the inter-onset interval). The inner representation (IRP) file has some similar properties to MIDI, having tracks and events. It consists of 17 tracks, the first containing the lyrics while the rest contain instrumental events. Each event is a 12-tuple, giving information such as: the start time and duration of the note, pitch, symbolic representation, interval from the previous note and chord information.

The decision to define the number of tracks in the IRP file to be exactly 17 was due to a confusion between tracks and channels in MIDI, which has 16 channels. A channel is a very different concept to a track; the MIDI file format is explained in more detail in Section 2.13. Storing both the exact pitches of the notes as well as the pitch intervals should not be required as this information can easily be derived at run time with little performance degradation (probably no more than reading the 12-tuple itself). It is stored to simplify any programs using the representation as the intervals to not need to be recalculated. The IRP was not referred to again in the rest of the paper, preferring instead to use the two-dimensional relative encoding.

Two-dimensional relative encoding is a representation which consists of a pair of components, relative pitch and duration, for each note. The method for matching the melody was dependent on where the query came from. Melodic pitch contours were only used when large errors were expected in the intervals. When a more reliable query was given, quantized partially overlapping intervals (QPI) were used. QPI builds on the idea of defining pitch directions as being small, medium and large. The category overlaps are an attempt to make the retrieval process more tolerant of errors but appears to result simply in an eleven-character alphabet.

The indexing structure used was a suffix-trie, a structure developed for string pattern matching. This is a tree like structure which stores each suffix of a word, e.g. “Hello” would be stored as “Hello”, “ello”, “llo”, “lo” and “o”. It is very fast for exact matching ($O(m)$ where m is the length of the query) but the space requirement is not good; it grows quadratically with the size of the database. To reduce the space requirement, only the top part of the tree, up to a certain depth, was stored - effectively using *n*-grams. Lemström and Laine did not specify the depth used.

Approximate matching was limited to errors in relative pitch, something which is not an issue with gross melodic pitch contours. However, it is possible to implement an improved matching algorithm which would allow for insertion and deletion of symbols. No timing information was given, so it is difficult to determine the suitability of this approach for an interactive system. However, due to the comprehensive indexing, one would imagine it to be superior to the systems developed by Ghias *et al* and McNab *et al*.

The requirements for the lengths of the query keys for each representation were considered using a database of 154 pieces of music that were “automatic compositions of Pauli Laine, folk-songs, etc.” They showed that the QPI representation returns optimal results, where optimal is the response given by exact intervals, with shorter keys than gross pitch contours (7 characters instead of 10 characters).

2.7.5 Meldex

Rodger McNab *et al* developed a system which searched a database of folk songs from a sung query [McNab96]. Transcription was performed by a system called MT (Melody Transcription), which accounted for the user’s gradually changing tuning in an adaptive mode [McNab95]. Searches were carried out using a choice of: an exact match of pitch; pitch and rhythm; pitch contour or contour and rhythm. Approximate matches were possible for pitch and rhythm or contour and rhythm.

A large database of 9,600 folk tunes was used to test the system and, using exact matching of rhythm and pitch, identified tunes with very few mismatches. However, the system only matched the beginning of tunes. The task of searching was further simplified due to the folk songs being single track and monophonic (only one note playing at any given time).

A complex system for transcribing acoustic input was employed. This results in an unnecessary processing overhead when not matching using exact pitch intervals, as a simpler algorithm is likely to track pitch accurately. The exact note sung is not important when using the contour, yet time was taken processing it. However, no indication of the speed of the system was given. They noted that a major issue would be the development of approximate string matching algorithms which avoid a computational explosion as the size of the database increases.

The system was given a web interface and named Meldex. It forms the basis of the music library functionality at the University of Waikato’s New Zealand Digital Library. In addition to the melodic search facility, which operates on the folk music collection, they employ an Internet spider to locate MIDI files on the Internet and then index any text contained within the file. This index can then be used to locate files by use of a text query. While queries to the MIDI text search can not contain melodic content, a larger number of files are indexed than in the folk music collection (over 100,000).

2.7.6 Melodiscov

Melodiscov stands for Melody Discovery and is a melodic search system that takes an audio query [Rolland98]. It is being developed at LIP6, Paris, as a WYHIWYG system (What You Hum Is What You Get). They claim that the audio query is transcribed with over 90% accuracy even when singing lyrics: many of the transcription components of similar work do not support singing of words. The search component uses several levels of representation (called descriptions) of the music that are used to describe the properties that are derived from the raw sequence of notes. There are three levels of description: individual, local and global. Information at the individual level concerns a single note or rest (e.g. specifying the duration and pitch). Local descriptions contain information on a segment, or musical phrase (e.g. that the passage consists of notes that are ascending in pitch). General information about the entire piece, such as the key or the average pitch, is given at the global level.

Each description may be assigned a weight for similarity matching, so that differences in important areas of the music effect the matching process more. These weights can be specified by the user, so allowing different viewpoints on what makes one piece of music similar to another.

2.7.7 MusiFind

The MusiFind project used an indexed *n*-gram database for music retrieval [Downie99]. The use of *n*-grams is a common technique for breaking up long sequences with no natural boundaries into manageable units. These units can then be used in algorithms and data structures that are traditionally only suitable for dealing with words. A window of *n* characters is slid along the length of a sequence to produce a list of *n*-grams.

The MusiFind project evaluated the effect of varying the size of the *n*-gram window (i.e. the value of *n*) for 3, 4, 5, and 6 characters. The number of pitch classes required was also investigated. Representations using 3, 7, 12, 14 and 23 pitch classes were considered. Only a subset of these was actually tried as it was a requirement for the hash table to fit into the memory of a personal computer contemporary at the time (i.e. one megabyte).

Stephen Downie’s work led on from the MusiFind project [Downie99], also using *n*-grams and then performing a full informetric analysis of the representations. He claimed that a representation with only three pitch classes (i.e. the gross melodic contour) is ineffective for retrieval as there is not enough information to differentiate one song from another. This is probably true for the values of *n* that were tried as they were relatively small. The claim implies a requirement that the number of notes given in the query is relatively small, compared to the length of the piece contained within the database.

Downie considers applying the concept of ‘stop words’ to musical *n*-grams. Stop words are used in many text retrieval engines to remove words from queries and indexed documents that have little resolving power (i.e. they are common to most documents). Downie explains that stop words exist in the English language due to the highly skewed frequency distribution of words. He claims that the distribution of *n*-grams is relatively equal and so stop words should have little, if any, effect. This is not proven experimentally.

The research showed that very little accuracy was lost by not verifying that the entire query is matched in files suggested by an index-only search. The built-in error tolerance of a representation with fewer interval classes does not result in greatly improved recall or accuracy. Instead, error tolerance is better served by a combination of a small *n* and a long query, where the *n*-gramming process provides adequate error correction.

2.7.8 MUSIR

In MUSIR, Salosaari and Jarvelin considered a retrieval model for music [Salosaari98]. MUSIR also used *n*-grams but with two, three and four units per *n*-gram. They considered the use of pitch interval and duration to retrieve themes from one piece of music, a fugue by J.S. Bach containing nine themes. Two queries were used in their experiment, one consisting of ten notes and the second adding another six to the first (i.e. the second query has sixteen notes).

This is a very small database, perhaps the smallest possible, but does highlight some interesting issues. They found that, while *n*-grams of three and four returned fewer matches, longer *n*-grams returned fewer relevant themes. This indicates, according to the paper, that longer *n*-grams risk not matching relevant documents. Errors in the query are accounted for by the n-gramming procedure, as is the case with MusiFind. Salosaari and Jarvelin note that the choice of *n*-gram size has an effect on the ability of the system to account for erroneous queries. A longer *n*-gram will require longer queries to match successfully allowing for errors, as the error will be present in more *n*-grams. Shorter *n*-grams will better support notes that are out of sequence, as the order is not noted when retrieving and short sequences are more likely to be out of order than long ones.

2.7.9 Query by humming

Ghias *et al* also utilised gross melodic pitch contours, but with ‘S’ representing repetition (or Same), for a database supporting audio queries [Ghias95]. The test database comprised a collection of all the parts (melody and otherwise) from 183 songs. They noted that 90% of these songs were retrieved with 10-12 pitch transitions. If the list of results was too large, the user could perform a new query on a restricted search list consisting of those songs just retrieved.

While the system allowed searching on any part of a song, and MIDI files are both polyphonic and multi track, it did have some problems. Like Meldex, the pitch tracking algorithm (using Matlab) was very complex and was the slowest part of the system. Searching the database would slow considerably as more songs are added and the sample database was too small to base any statistical analysis on. There was also a lot of unwanted information in the database, as it contained a lot of information unrelated to the melody (i.e. only drum / rhythm tracks were omitted).

2.7.10 Query by Pitch Dynamics

Query by Pitch Dynamics (QPD) was a system developed by the Link Group at Carnegie-Mellon University [Beeferman97]. It was part of the *by-content music indexing project* which aimed to explore automatic indexing and retrieval of all formats of music. This included more than just a tool to retrieve a MIDI file that contained some musical query. They also wanted to provide support for identifying songs that are similar to a query song and to extend this to automatic clustering of similar songs.

They invented p-structures which collapse a musical score to remove pitches that are not used. This retained as much information about the relationships of the notes that is possible without storing the relative pitch. Each distinct pitch in a sequence is given a rank, effectively removing unused pitches from interval measurements. Figure 2.4 shows an example graphical representation of (a) notes on a stave and (b) the corresponding p-structure. The example would be written as [2, 3, 1, 2].

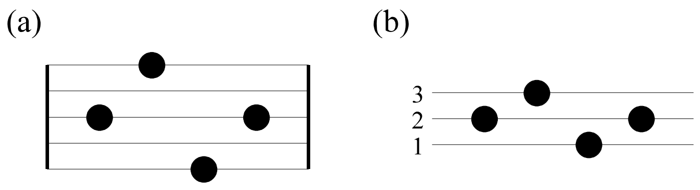


Figure 2.4 - An example graphical representation of (a) notes on a stave and (b) the corresponding p-structure

The p-structure is deduced for a window that is slid across the song and then mapped into a feature vector. Standard high-dimensionality storage and retrieval structures can be used, R-trees in this case. These can locate nearest neighbours to a query, which allows a very direct ranking strategy to be used. The expressiveness of p-structures is compared with that of melodic contours in Section 3.6.2 of this thesis.

On average, queries took 2.9 seconds to locate the 10 nearest files in a database of 1022

MIDI files. Attempts were made to deduce which track in the file was the melody by using some simple heuristics. Firstly, tracks that use less that five distinct notes were assumed to be uninteresting, and so not added. Secondly, the track name was searched for the word "melody". If a track contained that name, then the other tracks in the file were removed and the matching track used in preference.

The web interface also supported searching by artist or title and by MIDI instrument content. The last one is interesting as it allows queries such as "find all songs with more than 30% of music being on acoustic piano".

2.7.11 Themefinder

Themefinder is an interface into a collection of over 2000 monophonic classical themes [Kornstadt98]. It is implemented as a web interface to the Humdrum toolkit described above. A full complement of meta-data is also used, which includes the composer, genre, key and meter. The monophonic representations may be searched by using one of several music representations: exact pitch, pitch class, musical interval, semitone interval, scale degree, gross contour or refined contour.

2.7.12 Tuneserver

Lutz Prechelt and Rainer Typke at the University of Karlsruhe have written a web-based interface that uses a Java applet to pitch track a whistled query [Prechelt99]. The system then utilises gross melodic pitch contours to search a database consisting of the pitch contours given in the book by Denys Parsons [Parsons75]. It contains 10,370 classical themes and more than 5000 popular music songs written between 1920 and 1975. It also has a list of national anthems.

The system also facilitates bypassing the pitch tracking and entering the pitch contour directly. There are advanced search options that allow a limited amount of filtering based on the composer and title.

2.7.13 Vega

Vega was a multimedia database system that supports content based retrieval. It was developed by Liu and Chen and is designed to index both video and music [Liu97], although the video indexing functionality is not considered here.

Three features were used to index melodic content. These were: melody, as represented in do-re-mi form; rhythm, note durations (given as a number of beats) and chord sequence, where the chord for each bar is noted. These features were stored in the database as strings. The database was ultimately queried using their own Media SQL, although they also developed a more user-friendly query tool.

The music search interface took several forms of query: MIDI keyboard; entering the notes on a staff and by tracking the input from a microphone. Pitch tracking could use either a zero-crossing algorithm or a discrete cosine transform (DCT) based one.

Queries could allow for transposition errors, drop-out and insertion errors in both pitch and rhythm. The paper suggests that the matching process compares each melody with the query melody, one at a time. The stored melody is converted into a directed graph to perform the matching. No timing information is given but one would imagine it being quite slow, given the use of directed graphs, for which comparison algorithms are complex, and the lack of an indexed lookup.

2.8 Approximate string matching

The problem of searching a database of melodic pitch contours can be considered as a case of simple text retrieval [Downie99]. The field of text retrieval, and the more relevant approximate text retrieval, is well researched. The field is comprehensively explained by Navarro [Navarro98]; only a brief overview and details of information relevant to this work are presented here.

Developments in the area of text retrieval allowing errors were first used in applications such as biology and comparing DNA sequences. Researchers wanted to know how similar two sequences were, and what they had in common. DNA sequences can be represented as very long strings consisting of a limited alphabet (i.e. A, C, G, T). Finding occurrences of a shorter DNA sequence, allowing for some errors, is very similar to the retrieval of pitch contours.

In approximate string matching, there are two common types of search: searching with *k*-

mismatches, where only substitutions in the text are considered, and searching with *k*-differences, where insertions and deletions are also catered for. Whatever the type of search, the scenario will be one of two types and will have a large effect on choosing an appropriate search algorithm: the search may be on-line or it may be indexed.

In the former case, all documents in the system are searched sequentially for matches. This approach is required when space or time does not allow the documents to be pre-processed. Agrep is a Unix tool developed by Wu and Manber which supports fast online search [Wu92].

The second case pre-processes the documents, building an index, to improve subsequent approximate searches on those documents. A document may be indexed at the word level, where the exact location of each word is stored in the index, in blocks or as a whole document. Word-level indices tend to require more space than document-level ones but usually support more complex queries.

2.8.1 Query expansion

Exact search indices are well-developed structures in computer science. These allow fast lookup into large databases, sometimes in constant time. These data structures can be used in an approximate matching scenario by emulating errors in the query, so that exact lookup may be performed several times to locate inexact search terms.

This is only effective for a small number of errors in the query and requires a reasonable length query, or else there is a danger of matching the entire database. For example, a query of “Hello” and allowing for three errors requires that only two of the letters need to match, which is likely to be the case for many entries. In fact, the size of the expanded query set, for an alphabetic query that is not case sensitive, will be greater than 26^3 (= 17,576). This example figure is actually for the query of “Hello” where the first two characters must match.

2.9 Applying standard text retrieval techniques to music

It is common practice for text retrieval systems to make use of a stop list of the most frequently occurring words in a set of documents. These words are ignored by the system when both indexing a document and processing queries. This works well for English language documents as English adheres to Zipf’s law which states that the frequency of a word multiplied by its rank is a constant. This results in some words having a lot less resolving power than others, as they occur in most documents.

There may be clips of music which are more popular than others. Knowing clips that are the musical equivalents of ‘small’ words in English, such as ‘a’, ‘the’ and ‘it’, could be used to reduce the weight they carry when calculating a score for a match. It might be worth removing them from the query completely if the clip does not discriminate much. It should be noted that this is unlikely to have any grounding in the theory of music.

Chi Lap Yip found that, for certain lengths of *n*-gram, musical *n*-grams behave statistically like English words [Yip99]. Downie performed a similar analysis but came to a different conclusion, finding that the rank/probability of interval *n*-grams is skewed when compared with English test. It is worth noting that the sizes of *n*-grams he considered were smaller than those used by Yip.

2.10 Open hypermedia

Many approaches to hypermedia have been developed. Some of these, such as the Amsterdam hypermedia model [Hardman94], specifically consider audio. One which is relevant to content based navigation, and so considered here, is the open hypermedia philosophy [Hall94], as used by Microcosm [Davis92] [Fountain90] and the DLS [Carr95] [DeRoure96].

2.10.1 Key aspects of the model

There are two key aspects to the open hypermedia model. Firstly, information about links between documents is stored separately from the documents themselves (in contrast to common practice with HTML documents). This information is maintained in link databases (linkbases) and, by selecting different linkbases, the user can obtain different views of the documents. Access to linkbases may be abstracted into a link service, which is a network service supporting the resolution of links.

Secondly, complex processing and reasoning about the links may be performed by the link service. Taking the *generic* link, as found in Microcosm, as an example. When the user requests available links, the linkbases are queried with either the location in the document or with the content at that location in the document. This means it is possible to link from anywhere in a document. In this case, the processing involves searching the database of links for suitable matches based on the content or position of the source anchor. For example, a link may be authored such that any occurrence of the phrase "University of Southampton" is associated with the main page on the world wide web for the University.

Using an open hypermedia system, it is possible to author a link from a phrase to another document in one step. Users of a system may also be authors of links, even if they do not have access to the documents on which those links are based. This is possible because the links are held separately from the documents. Most existing systems only allow linking from text documents, very few have support for audio and none allow linking from musical constructs.

The approach supports link resolution that is more complex than matching simple phrases. The context of the source phrase in the document could be used to deduce the appropriate definition of bear and only return links relating to grizzly bears if the document is discussing the animal, as opposed to the verb.

2.10.2 A standard communication language

The Open Hypermedia Systems Working Group (OHSWG) has developed the Open Hypermedia Protocol (OHP) [Davis96]. This was aimed at providing a standard data model and socket protocol for the communication of link information between open hypermedia systems. The common features of existing open hypermedia systems were identified and abstracted into a simple protocol. An existing system would need only to write a 'shim', a software interface between OHP messages and the system, to be able to interoperate with other hypermedia systems.

The first version of the protocol was presented by Davis *et al* and was based on the message format used by the Microcosm system, a list of tag/value pairs. Due to the considerable time being expended on the development of parsers for this format, the protocol was modified to become XML compliant. This was to take advantage of the wide availability of XML tools and parsers for a variety of different platforms.

The original aim of the protocol was for interoperability between clients and servers. As the scope was expanded to encompass the research interests of all members of the OHSWG, it was realised that a single protocol could not be all things to all people. As a result of this, the standard was split into several domains, representing the different domains of hypermedia. The standard was renamed to OHP-Navigational (OHP-Nav) and reduced in scope.

2.10.3 A common data model

Members of the group who are based in the University of Southampton have moved away from defining a protocol for communication and towards developing an abstract data model of hypermedia. It is hoped that, if hypermedia systems implement the same data model, the protocol for communicating link information will become a secondary issue. This is the goal of the Fundamental Open Hypermedia Model (FOHM) [Millard2000].

Formal models were developed of the different domains of hypermedia (navigational, spatial and taxonomic). The concepts that are fundamental to all models form the basis of FOHM. While an object in one domain may define more attributes than are common to all domains, some attributes may be used in a separate domain.

2.11 Content based navigation

The Multimedia Architecture for Video, Image and Sound (MAVIS) project [Lewis96a] [Lewis96b] was an open hypermedia system which allows links to be based on content from pictures, sounds and video clips. Although primarily concerned with image content, it did support audio as a proof of concept. The features used to represent audio were frequency spectra (FFT’s). These could be approximately matched with an audio query to find related files.

The application of FFT’s is of limited use, as matches using low level representations are more prone to return irrelevant documents. This is because low level data is not suitable for performing high level comparisons. For example, a short piece of music played on two different instruments, perhaps an oboe and a flute, will have quite different frequency spectra yet might be expected to match on the tune being played. A straight match of the spectra can only match the tune played on the same instrument, as long as it is played in the same way. The abilities and shortcomings of using this representation for comparing audio are best illustrated by David Gibson and his “Name That Clip” software (see above).

Fourier transforms are better suited for instances where the timbre, the relationships between the harmonic frequencies, is important and enough to distinguish between the different samples. The retrieval of digital audio samples, as discussed above, is a good example of this.

Navigation is easily confused with retrieval, as illustrated by Hirata *et al* whose media-based navigation system (Miyabi) allows users to use “the same type of media for a query as the media they want to retrieve” [Hirata93]. This is not navigation but retrieval, as only media similar to, or containing, the query may be retrieved. Navigation should also allow media relating to the query to be found.

2.12 Identifying irrelevant data

The performance of most matching techniques could be improved by discarding content which is not likely to be the subject of a search; so less material is indexed. It should be noted that not all of these algorithms will naturally apply to navigation, due to its highly selective use of anchors. That is, only relevant information is stored in manually generated linkbases by default. In this situation, these techniques could be used in reverse to ignore queries referring to content that would not have been indexed.

2.12.1 Melody extraction

The ability to identify the role of each instrumental line in a polyphonic piece of music is very useful. The typical query on a music database is likely to be the main melody line, so identifying this is of particular interest.

Uitdenbogerd and Zobel [Uitdenbogerd98] tried four algorithms for producing a monophonic melody from a polyphonic MIDI file. The selection of algorithms was based on music perception research which suggests that the sequence of notes with the highest pitch is likely to be perceived as the melody, unless there is a very interesting, i.e. nonrepetitive, underlying sequence. The four algorithms they tried are given in the Table 2.

Table 2 - Melody extraction methods evaluated by Uitdenbogerd

Name	**Method**
all-mono	Combine all the MIDI channels into one and pick the highest note playing at any given time
entropy-channel	Use a simple measure to determine how interesting each channel is and keep the most interesting one
entropy-part	Split the notes into parts, using heuristics, and pick the most interesting one
top-channel	Keep the MIDI channel with the highest average pitch then take the highest notes, as in all-mono

A small user trial suggested that the all-mono algorithm performed best, giving acceptable results for many cases. Naturally, this approach will fail when the melody is not the highest pitch being played.

The problem they have tried to solve is not strictly the same as that given as the motivation in this section. Rather, given some polyphonic music, they convert it into a monophonic representation such that it may be successfully indexed and retrieved. This is thought necessary due to the majority of systems and indexed representations being monophonic. Kjell Lemström and Jorma Tarhio remove this limitation by modifying an on-line search algorithm to locate monophonic sequences in polyphonic music [Lemstrom2000]. That is, a single note in a query will match that same note if it occurs in a chord.

2.12.2 Classification systems

Standard classification methods are applied in Chapter 4 to determine the melody line. A number of approaches to classification are reviewed here. At a high level, all classification algorithms work in a similar way.

Classifiers require a set of pre-classified samples by which their performance can be evaluated, and many require that same data to implement or train the classifier. A set of features, referred to as a feature vector, is extracted from each sample. The choice of features used will depend largely on the type of media being classified but might have to be tailored for each classifier.

All learning classifiers must be trained before they can be used. The accuracy can then be evaluated against the validation set. It is common for the set of samples used for validation to be separate from the training set. This prevents classifiers that simply memorise the training data from giving the false impression that they are 100% accurate; instead they are realistically evaluated.

2.12.2.a Ad-hoc classifiers

Possibly the simplest approach to classification, at least in terms of concept, is ad-hoc. This directly codes the knowledge of a person familiar with media into the classifier.

This can be effective for media types which follow clear and simple rules. Trying to differentiate between pictures of sea and grass can be simply coded. Differentiating between pictures of sea and sky is likely to be more difficult.

2.12.2.2 K-nearest neighbour

One method of classification is to find the pre-classified feature vector which best matches the one to be classified. The classification of the query vector is assumed to be the same as that of the nearest match. In a *k*-nearest neighbour classifier (K-NN), the full representation of each feature vector in the training set is stored, along with the classification for each vector. Given a vector to classify, the stored vectors are searched to find the *k* that are closest. The classification of the closest vector is returned as the classification of the query vector. Closeness is measured as the Euclidean distance between the vectors. The nearest neighbour and *k*-nearest neighbour algorithms are both established techniques for multimedia retrieval [Chen].

This approach has the advantage that more pre-classified samples can be easily added to the set stored by the classifier. No information is lost, as all the data is kept, which makes this approach highly effective. This has the consequence of having relatively high space requirements and can be slow to classify.

2.12.2.3 Neural networks

Neural networks are normally faster than *k*-nearest neighbour classifiers, as evaluation requires only a few operations on relatively small (compared to the training set size) matrices. One can consider neural networks to be an approximation of the *k*-nearest neighbour method. As such, they are not generally as accurate.

Neural networks are designed to model some aspects of the human brain. A network has a number of input nodes which are cross-connected to a number of output nodes, often via some hidden nodes (see Figure 2.5). These connections are weighted so that some nodes are 'more connected' than others. This affects the flow of data from the input nodes to the outputs. The network must be trained, to learn appropriate weights, before it can be used. This involves presenting a set of inputs while also giving the desired output for that case. The internal connections of the network are modified towards matching the example output, so that it should learn valid outputs for any given input.

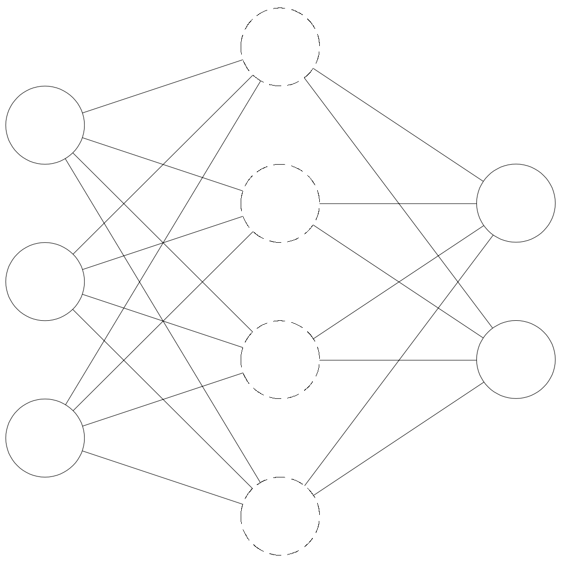


Figure 2.5 - A simple neural network with three input nodes (left), four hidden nodes (centre) and two output nodes (right)

The mathematical theory and internal workings of neural networks need not be understood to be able to use them. They can be thought of as a black box, although it is important to know that there are a number of parameters controlling the configuration and learning abilities of the network. These must be tuned for optimal performance; there are tools available to aid this tuning.

2.12.3 Analysing the structure of music

Although a musical performance is linear, there is an underlying structure to the music. Analysing this structure could help identify key themes in the music, such as the ‘hooks’ (the easily memorable clips) in popular music or solos in classical music. Once the structure has been deduced, this could itself be used as a form of linkbase, e.g. navigating between all hooks in a piece.

Deutsch worked on isolating solo performances from digital audio [Deutsch98]. This used the principle of psycho-acoustic masking, which identifies and removes sounds that are not prominent in a signal. A common application of psycho-acoustic masking is in layer 3 of the MPEG-1 standard, which uses it to compress audio. The standard compression ratio (12:1) produces high quality audio, bearing a high resemblance to the original audio. Deutsch increased this ratio, to start removing sounds that would be noticed (although not much) by the listener; this is called overmasking. The ratio was increased to the extent that only the prominent part of the audio signal remained. He found that, in the case of classical music, this was usually the solo part. It is important to realise that this is very genre specific though, as solos in classical music are usually the loudest in an audio mix. Other genres, such a pop music, do not usually follow such rules.

Arbee Chen *et al* has investigated methods for the automatic discovery of repeating patterns in music. It is accepted in musicology research that musical themes are usually repeated throughout a piece of music. They developed an algorithm that used a correlative matrix to locate musical sequences that were repeated exactly throughout a piece [Chen98b]. Empirical tests were performed on a collection of classical works. The average execution time was ten seconds for a piece with 1000 notes. A new data structure, called an RP-Tree, was developed to reduce the execution time to under four seconds for a piece with 1000 notes [Chen99]. The improved algorithm also removes trivial repetitions in an attempt to find the segments that are most likely to be the theme of the music. Extraction of repeating sequences has also been examined by Tseng *et al* in their music retrieval system [Tseng99].

The structure of music has a further use for content based navigation. One of the problems with an interactive paradigm, such as Open Hypermedia, is automatically determining the size of the selection in the media which is of interest.

An easily digestible example is of someone listening to a news broadcast. A suitable user interface might consist of a single button on the listening device that the user presses when they wish to find out more about a story. The structure of the broadcast needs to be determined for effective linking. Where are the boundaries of the current news item? In a similar way, the structure of music can be used to aid the navigation of the media by passing the whole of the current musical phrase to the navigation system.

Melucci and Orio segmented musical works into melodic surfaces, short musical phrases, and indexed these phrases [Melucci99]. The segmentation algorithm was based on the Local Boundary Detection Model (LBDM) which was proposed by Cambouropoulos. The model works on the assumption that a listener will perceive a change of phrase when they hear some changes in the relationships between the notes. The transition between each note in a piece of music is assigned a weight; higher weights suggest that a change in the melodic surface is likely. The weights are then analysed to find the largest values in the piece, which is where breaks are most likely to be located.

2.13 The MIDI file format

The Musical Instrument Digital Interface (MIDI) specification was originally designed as a transfer protocol by which electronic instruments could communicate with each other. It continues to be used in this way but has also been adapted as a file format. As this thesis often refers to MIDI files, a detailed description of them, and the MIDI transfer protocol, is given here.

The transfer protocol operates on a 31.25 Kbaud serial link. The link forms a daisy chain through all connected equipment, so that only two MIDI ports are required on each instrument to connect an entire studio. The MIDI protocol is optimised to take up as little space as possible on the link, and so reach the receiving equipment in a timely manner.

As all the instruments in a MIDI-enabled studio may be connected in a chain, MIDI has the concept of logical channels. There are 16 logical channels which are used to address individual instruments. For example, the drum machine may be set up to transmit and receive information on channel 10, while a keyboard may use channel 1. There are conventions that suggest the purpose for which some channels should be used; one such convention is called General MIDI.

A MIDI message, called an event, contains commands such as: note off; note on; after touch (modify the amplitude of a note currently playing); specify a different sound or system messages (also called real-time messages). Note on events specify both the pitch and the amplitude of the note to be played. The pitch is given as a number between 0 and 127, where 0 is defined as C0 and 127 as G10.

The MIDI file format uses the same basic principles as the transfer protocol. It is a Resource Interchange File Format (RIFF) compatible file, where the file is organised into chunks that may be parsed without knowing about MIDI (although very little is achieved by such a process). The first chunk is a header which identifies the type of file, the timing resolution of the file and how many tracks the file contains. A track can be thought of as a single recording take, which might be played in parallel with other tracks. The track concept is useful for musicians, as it allows parts to be managed individually. For example, the piano part may be recorded separately from the bass, and re-recorded if the first take was not good enough.

Each track is essentially stored as a binary dump of the MIDI events from the transfer protocol, along with a time stamp for each event. The only exceptions to this are some real-time (also called system) messages. One in particular is the MIDI reset message, which directs all receivers in the system to return to their power-on status. Clearly this is not a useful event to store in a file and is adapted to identify meta information. The meta event is used to name each track, note copyright information and lyrics. It is also used to modify the tempo and time signature of the file.

2.14 Summary

It can be seen that content based navigation of audio has not been the subject of much research, with only MAVIS supplying audio as a proof of concept rather than an aim. The next most important research is the similarity matching of music used in content based retrieval, as a CBN system will have to compare content at some point.

Both of the systems developed by McNab *et al* and Ghias *et al* have characteristics which are not well suited to the intended application in content based navigation. The former only indexes the first part of each piece, while all alignments need to be identified, and the scalability of the latter needs to be improved upon to ensure response times which are within bounds for an interactive content based navigation system.

Ghias *et al* suggested that three-way discrimination of pitch might be useful for finding a particular song among a private music collection, but that higher resolutions would probably be necessary for larger databases. Lemström used suffix-tries and effectively resolves music to 11 pitch classes. The indexing engine used suffix-tries which gave a fast response time but the size of the database would not scale well.

Content based retrieval and navigation have a lot in common. Both applications exhibit the same basic requirements, namely fast approximate matching of some query content against a large database of pre-processed content.

It is likely that the specific requirements may vary for each case. For example, the source of queries for retrieval may come from a number of different sources with inherent errors in the query (the nature of the errors may be dependent on the source). Navigation implies selection from an existing piece of music, so the query is more likely (than in retrieval) to be free from error, but the size of the query may well be different from the retrieval case. The anchors in a music linkbase are likely to be smaller than the complete songs used in retrieval but, assuming more than one link per song, may contain more targets. These factors could affect the fine tuning of a database for a particular purpose.

3 Retrieval of music

The similarity between content based retrieval and navigation (as explained in Section 2.6) results in the majority of this thesis considering the simpler case of retrieval. The application of retrieval techniques to navigation is considered in Chapter 5.

This chapter considers the selection of a representation and the design of a database structure to support the approximate retrieval of music. The different approaches to implementing the database are evaluated for speed, while the effectiveness of the database design is measured. It introduces a new pitch representation that helps improve the accuracy of the database.

3.1 The nature of a query

It is appropriate at this point to make some comments on the assumptions made about the queries submitted to the system. One of the likely applications for a music retrieval system is for a user to submit a query in order to locate a song. The query might be entered by a music student using a MIDI enabled device or by placing notes on a score in a notation system. Someone who is not a musician might sing or hum a piece of melody to the system. Whichever of these methods is used, one can assume that the query will not consist of the entire song (or else they are unlikely to require a retrieval system) and that the query may contain errors. The query is also likely to be quite short, perhaps under ten notes from a layman user. For example, the opening theme to Beethoven's fifth symphony is recognised by many people from only eight notes.

Taking a segment of a song not authored by the user as the query, e.g. a MIDI file or a score from a music archive, has another set of properties. In this case, the query is not as likely to contain errors, or at least not the same types of errors. This is due to the query not being dependent on human recall and because different methods of conversion from the physical to the digital domain are likely to be employed (e.g. Optical Music Recognition (OMR) for printed scores). The length of a query is likely to be longer than in the first case, due to not having to rely on how much the of the song the user can recall. The reasons for using an existing work as the query are also likely to be different from the ‘query by humming’ scenario. Instead of determining the name of the song that the user is singing, questions such as “where is this theme also used” are to be expected.

The retrieval of music as considered in this chapter has two purposes. Both of the scenarios are relevant. As previously discussed, retrieval can be used to locate the first documents in a browsing session. Retrieval can also be an aid when creating links for navigation. The second scenario is also of great interest, as it most closely matches the specifications of queries in a system used for navigation.

3.2 Compiling the music collection

Before considering the development of a system for retrieving music, it is a good idea to first obtain some music. The music will be required for both investigating requirements and for testing the resulting retrieval system. Only MIDI files are considered, as large numbers of these exist and they avoid the pitch tracking problems associated with other formats, such as MP3 files. This also clarifies the subject of any later evaluation, i.e. evaluating the ability of a system to retrieve digital audio files would be testing the pitch tracking function as well as the retrieval process under test.

The requirements for MIDI files in the collection were that they should be representative of those likely to be indexed by a ‘real’ retrieval database. This should ensure that they will also represent realistic source material for a content based navigation system. This means multi-track, polyphonic music. Multiple arrangements of the same songs are allowed, as they can be used to test the generality of the search procedure. Exact duplicates should be removed.

The test music collection was obtained from an FTP site on the Internet[5]. At the time the

[5] Trantor FTP site: ftp://cam031313.student.utwente.nl/pub/midi [10 February 1999]

files were copied, the site contained over ten thousand files. Filenames which only differed in capitalisation were not downloaded. The checksum of each file was compared and exact duplicates removed, although this did not remove multiple versions of a song. This further reduced the number of files by 500. Removing files which the database system could not use, either because they were single track MIDI files or because they were corrupt, left just over 7500 files for the experiments.

The collection contained over 30 million notes, averaging 120 notes per minute of each MIDI track and eight tracks per file. The average length of a piece of music in the database was three and a half minutes. This is considerably different to the folk music database used by both McNab *et al* and Downie, where the melodies were an average of 56.8 notes [McNab2000].

3.3 Choosing a representation

The representation chosen for this work was the gross melodic pitch contour, as it has some interesting properties and poses some intriguing questions. It is likely to be forgiving when used with human queries, as people are unlikely to recall melodies with 100% accuracy. Pitch contours are relatively vague; does this result in too many false matches? Or, conversely, does this built-in “fuzziness” result in allowing a straightforward exact match algorithm to have some limited approximate matching behaviour? An added advantage, at least for future work, is that the difficult problem of pitch tracking might be simplified if only up, down and repeat need to be transcribed.

3.4 Database types

It is obvious that fast retrieval will require more than a collection of text files and a standard linear text search utility such as the grep utility, especially as more files are added to the database. Is a specialised database required for optimum performance, or are there viable / suitable alternatives? Three possible database types are considered here. They were chosen as being representative of the selection that a software engineer is likely to choose between. One is specialised to the problem, another is a standard relational database while the last is a text indexing engine - given the previous observations that regard the process of matching pitch contours as being a case of text retrieval.

The pitch contour for a track is long, an average of 310 pitch directions per minute in the test material. To allow an efficient database lookup technique to be employed, each track is stored as a set of overlapping sub-contours; sub-contours are the atomic units of the database structure. The length of a sub-contour is referred to as the atomic length, *L*. To give an idea of a typical value, *L* = 12 for the database initially used to test the system. This is an important constant in the database structure as it defines the number of distinct atomic units that may be held in the database (which is referred to as the atomic resolution in this thesis).

$$\text{atomic resolution} = 3^{L}$$

This principle is applied regardless of the actual database implementation. It is a widely used technique; the sub-contours are often referred to as *n*-grams in related literature. Lemstrom notes that the ideal implementation would store all possible post-fixes of contour but that this results in huge storage requirements. It is also unlikely that user queries would take advantage of such rich indexing. That is, user queries are unlikely to be of an entire song, or a large majority of it.

To add a contour to an *n*-gram database, it must be split into sub-contours by sliding a window of *L* characters along its length. An example is shown below where *L* = 12.

```
The contour:  DURDRUURDRUDUR
Becomes:      DURDRUURDRUD
               URDRUURDRUDU
                RDRUURDRUDUR
```

3.4.1 Custom database

The aim of the custom database design was to facilitate the fast lookup of a node when a contour *n*-gram is the key. Using a perfect hashing function, i.e. one that maps different keys to different addresses, to index into a lookup table allows just this. As the hashing function is perfect, there is no need to compare the nodes found at that index, in fact, there is no need to store the key at all. The use of perfect hashing results in a larger space requirement but, with availability of cheap, high capacity, storage devices increasing, the speed benefit was considered to out-weigh this drawback.

The custom database is best described by explaining its operation. Occurrences of a sub-contour in the custom database are located by converting the query contour into a base 3 number (where U=0, D=1 and R=2). This gives the relevant index in a lookup table stored at the beginning of the database file. Each element of the table points to byte locations further in the file, identifying the start of the linked list of matches (see Figure 3.1). If the element contains the value zero then there were no matches for that sub-contour.

Each node in the linked list contains: the byte offset in the file to the next node; the id of the file the node relates to; and the number of times this contour occurs in the file. The file id is a unique number for which the corresponding file name can be found by using an associated name database. There is only one node stored per sub-contour per MIDI file which often produces a compression effect as this usually refers to more than one 'hit'. It should be noted that the sub-contour is not actually stored in the database, but rather implied by the position of the related data in the lookup table at the start of the file.

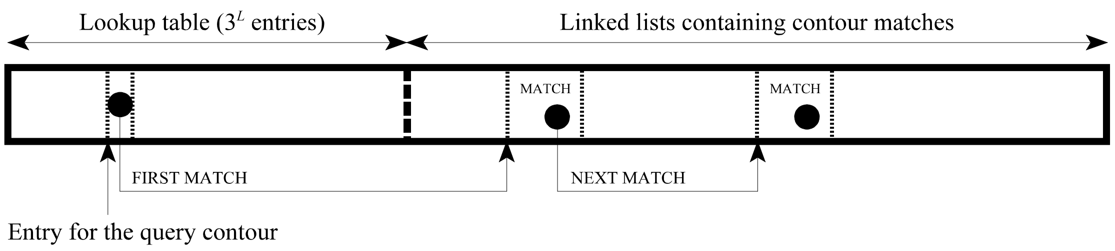

Lookup table (3^L entries)
Linked lists containing contour matches
MATCH
MATCH
FIRST MATCH
NEXT MATCH
Entry for the query contour

Figure 3.1 - A diagram showing the operation of the database

This database structure makes it possible to determine if there are any matches for a given query in constant time. Each subsequent access retrieves one match. However, this is a *lookup* technique and only works with exact matches. The lookup table can be considered as a hash table which uses an optimum hash function. This function maps each distinct, L length, pitch contour to a unique location in the lookup table. This allows for constant time lookup, as one disk access ensures relevant data is located, but has the disadvantage of limiting the number of distinct contours indexed. How limiting depends on the particular architecture used. On an Intel Pentium III system, this is a practical maximum of 16.

The limit is due to the amount of space required by the lookup table. For example, a file pointer is 32 bits wide on the development system, which allows files up to 4 Gbyte in size.

This means that each lookup table entry requires four bytes of disk space. So an atomic length of 3 would give a total of 27 distinct contours, that is 108 bytes for the lookup table. The table reduces the amount of addressable space left for the actual node entries (4 Gbyte less 108 bytes). This effect is not noticeable for small values of *L* but quickly becomes an issue as the index table grows exponentially with increasing atomic lengths. Figure 3.2 shows the maximum number of nodes that may be stored in a 4 gigabyte file - when a node occupies 12 bytes. It can be seen that the size of the lookup table has a negligible impact on the rest of the database where *L* <= 15. Over 1.44 gigabytes of disk space are required for the table where *L* = 18, which is the theoretical maximum for 32-bit file pointers. Figure 3.2 shows this more clearly, where it is shown that there is not enough space for the lookup table when *L* = 19 (the maximum number of nodes is negative).

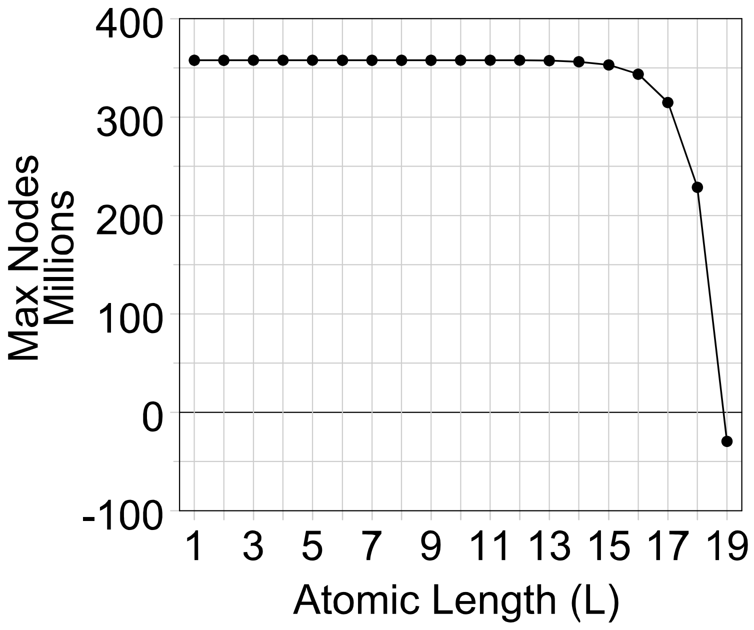


Figure 3.2 - A graph showing the effect of atomic length on the maximum number of nodes that may be stored in a database file

Keeping the lookup table as a separate file would not overcome this restriction entirely, as there would still be an upper limit of 18 pitch directions, although over 100 million additional nodes could be stored. Another disadvantage of using one-to-one hash tables is that space is quickly wasted when a higher value of *L* is used. This is due to the uneven distribution of pitch contour *n*-grams in music, so some may not occur at all in a collection.

3.4.2 General purpose database

A standard (table based) database is used as the back end. The native support for Paradox tables in the Borland Database Engine was the particular one used, although most databases should produce similar results (e.g. Microsoft Access). The method of indexing used by a particular database will be implementation dependent. It is likely to use a fast, general purpose, index. Hash tables using hash functions which do not map distinct indexed keys to unique values is one such structure. This approach has the advantage of allowing greater length *n*-grams to be used as a direct result of this generality. The indexing algorithm might be tuned for natural language text, where more than three characters of the alphabet are used. This may lead to a decrease in performance when used to index pitch contours.

3.4.3 Exact text retrieval engine

As this work represents pitch contours as simple ASCII text, an off-the-shelf text indexing database was used for comparison purposes. This was the Onix library[6] from Lextek International. The downside of using this particular library was that the index must be built in one pass for it to work efficiently. There is a limit on the number of updates possible to the index, before having to start again, of 256. This is not likely to be the case with all text indexing and retrieval libraries. On the positive side, the atomic length is not limited to 18, like the custom file format is, and it claims to be able to retrieve from the index with a single disk access (on average).

Text retrieval engines have similar properties to table based databases. They are effectively hard-wired tables that model the relationship (word $\rightarrow$ locations). This specialisation may result in a slight speed increase over standard database systems.

[6]Onix is available from Lextek International, see http://www.lextek.com for more information. It was generously donated by Art Pollard, the author of Onix, for these experiments.

3.5 Query expansion

The design of a near instantaneous system which needs only to locate content that exactly matches the query is relatively straightforward. All of the database systems considered here are designed for exact matching. Whilst relational database systems may provide limited approximate matching through the SQL 'LIKE' command, this is geared towards wildcard matching and not for *k*-mismatches.

Performing an approximate match using the database structure alone would require the entire database to be searched. Obviously this does not scale to large databases. The notion of approximate queries, as opposed to approximate matches, is used to solve this. Possible errors in the query are emulated, the database searched for the resultant list of near matches and the results collated.

This technique is related to the process of query expansion as is commonly used in database systems to improve recall. In this case, it is common to add terms to the original query with a view to locating related documents. A thesaurus may be used to add the search term 'audio' to a query containing the term 'sound'. Documents found to contain these additional terms will not usually have the same weighting as those containing the original terms.

To match a query, the set of contours that are within a user-defined distance, *d*, of the query is produced. The distance between two contours is given by a string metric that quantifies the minimum cost of transforming one contour into another. Cost weights may be assigned to the individual editing operations involved in such transformations, namely symbol substitution, insertion, and deletion.

The metric used may depend on the source of the query, which could be extracted from a MIDI file or from a digital audio file with monophonic or polyphonic content. It may come from original source material or via the tune recall skills of the user. This system employs the Levenshtein distance [Levenshtein65], where each transformation has a cost weighting of 1, but in each case certain types of error may be more common so this weighting may not be the most appropriate. For a query by humming system, more research into tune recall is needed to identify suitable weights. It may be possible to establish appropriate weights automatically for the metric by learning from a reference set of data [Ristad69].

Two methods of query expansion are considered here. The first, described in the following section, is an algorithm for achieving the query expansion as explained above. The second, given in Section 3.5.4.2, uses an alternative approach to cater for possible errors in a long query. Choosing the most suitable expansion method requires calculating the number of additional search terms that are introduced.

3.5.1 Tree fuzzy query expansion

A brute force approach to collating the expanded query set is to iterate sequentially through all possible contours of length *L*. The Levenshtein distance between the query and each possible contour is evaluated. Contours that are found to be within the distance limit are added to the expanded query set. Additional and missing pitch directions are catered for by the string distance metric which considers these errors to have a weight of one. In this way, a distance limit of one will allow pitch direction to be inserted, omitted or replaced.

It is possible to implement a ‘brute force’ algorithm which uses a tree structure to ‘visit’ each contour. Consider a tertiary tree which has a depth equal to the atomic length, *L*. Each node contains a contour. The contour at the root node is empty. The tree branches three ways at each node, appending a pitch direction to the contour at each level. The set of leaf nodes is the set of all possible contours (see Figure 3.3).

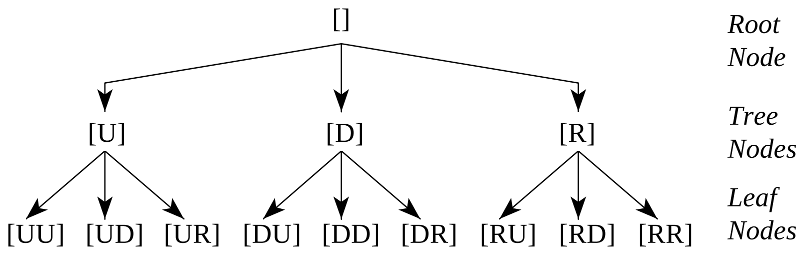


Figure 3.3 - The complete query expansion tree, where $L = 2$

A simple tree search would traverse the entire tree, comparing the contour at each leaf node with the query. This algorithm can be refined by comparing the contour at each node with a prefix of the query contour. If the distance limit is exceeded then there is no point in traversing the rest of that tree, as the error can only increase. This allows large sub-trees to be removed, reducing the number of leaf nodes that need to be compared.

There is a further improvement which can be made. Currently each node contour is compared. There are two situations where the distance between a node and the query is irrelevant:

1. If the length of the contour at a tree node is less than the number of errors allowed, then it must be within tolerance;
2. If the height of a sub-tree is less than the remaining number of errors allowed then all leaf nodes of the sub-tree will be within tolerance and no node comparisons need be made.

3.5.1.1 Evaluation of tree fuzzy matching

The speed of the query expansion was evaluated by expanding as many randomly generated queries as possible in one hour. The results for an atomic length of 12 are given in Figure 3.4. This compares favourably with the brute force approach of trying each possible contour in turn, which took 74 seconds on the same machine. It is interesting to note that even when all contours start to match (when *d* >= 10) the execution time is faster than using brute force, as very little comparison is performed.

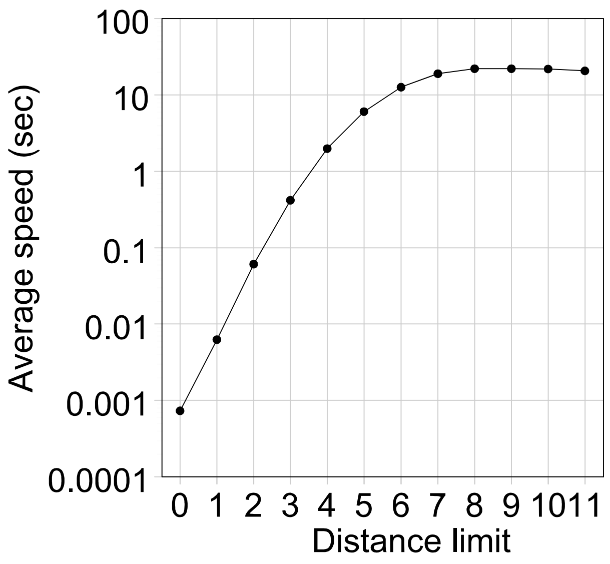


Figure 3.4 - A graph showing the average execution speed of the tree-based query expansion, where $L = 12$

3.5.1.2 Calculating the size of the expanded query

The number of contours with *d*-mismatches can not be calculated accurately in advance, due to it being highly dependent on the query. A general formula for the worst case can be determined.

First consider the case with no errors, i.e. there can only be one contour. The set allowing for one error contains contours where each pitch direction in the query has been replaced with the other possible characters (i.e. two, for a three-character alphabet). It also contains the exactly matching contour, so $n = 1 + 2L$ (when L is also the size of the query).

The case for $d = 3$ can be considered an extension of $d = 2$. Each emulated error in the previously mentioned contours will in turn will be fixed while an additional error is applied. This leaves one less position to try (L-1), so the result of $d = 2$ is multiplied by 2 for (L-1).

This leads to the general form: $n = 1 + \left((2L)(2(L-1))(2(L-2))\ldots\left(2\left(L-(d-1)\right)\right)\right)$, where $d < L$. This can be better expressed as the following equation.

$$n = 1 + 2^d \left[\frac{L!}{(L-d-1)!}\right]$$

To check the validity of this model, the number of distinct *n*-grams in the expanded sets was noted at the same time as testing the execution speed of the tree-fuzzy expansion. The actual minimum, maximum and average set sizes were plotted, along with the estimate given by the first equation (Figure 3.5). This showed the equation to be a relatively poor model of the actual sets. This is because following the process described above will duplicate contours during the expansion. A better approach is to use the pick, or permutations, function which is defined as:

$$n\mathrm{P}r = \frac{n!}{(n-r)!}$$

This has some visible similarities to the original equation. The permutations function can be used to determine the number of substitutions that are introduced for each increment in the number of errors emulated. The following model (referred to as equation 2) was implemented and plotted alongside the original equation (Figure 3.5). This also did not produce perfect estimates but more closely matched the observed sizes. Due to the complex interaction of substitutions, insertions and deletions, it is likely that applying a curve fitting algorithm to the observed data will be the only way to successfully model the problem. Equation 2 should be adequate for the purposes of this thesis, as the intended application of this metric only requires an approximation (see Section 3.5.4).

$$n = 1 + \sum_{i=d}^{i=1} 2(q \, \mathrm{P} \, i)$$

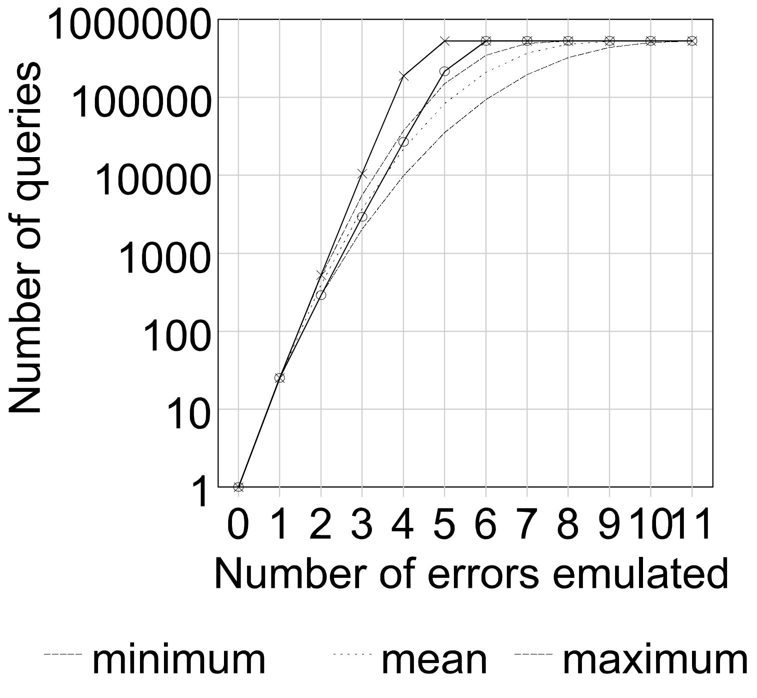


Figure 3.5 - A graph showing the relationship between the calculated expansion set sizes and actual set sizes

3.5.2 Collating the results

Tune retrieval can be compared with text retrieval; for example, the number of occurrences of a contour in a MIDI file is significant. However, it is perhaps more unusual in text retrieval system for the query to be so prone to error; as noted above, contour matching is sometimes more akin to spell-checking; a more intelligent form of query expansion, e.g. returning the ten most likely pitch contours, could be considered a spelling checker for contours. This means that the policy for ranking results must be different, giving priority to a large number of hits with one error over a single precise hit.

A score is calculated for each file by summing the number of times a contour in the expanded query set occurs in the file. Each hit is inversely weighted by the distance between the original query and the contour matched. The inverse weighting ensures that less importance is given to contours matched which are increasingly different to the original query. The search results are then sorted in order of their score, highest first.

The system favours matching on large MIDI files, as they contain more contours, so the scores are higher. To overcome this, the contour count could be normalised upon adding a file to the database. All counts would be expressed as a value indicating the percentage of the file that each sub-contour represented. For example, a sub-contour which occurs 100 times in a file containing 1000 (indistinct) sub-contours in total would have a contour count indicating the fact that it represents 10% of that file. This is a topic for future work.

3.5.3 Handling short queries

Tree-fuzzy query expansion is fine for queries of the same length as the atomic length of the database. Queries less than the atomic length are easily catered for by trying all possible endings for all of the expanded smaller query. If the query is much smaller than the atomic length then this approach can incur a considerable overhead, as the expanded set can become quite large. The database should be rebuilt with a smaller atomic length if queries are consistently too small. Alternatively, multiple databases could be built if the lengths of queries are inconsistent, querying the database whose atomic length is closest to the query length.

An obscure side-effect of sliding a window along the length of a contour to add it to the database is that the last L - 1 characters can not easily be located in the database. For example, take a contour of 13 pitch directions (a) and let L = 12. The two sub-contours resulting from sliding the window will be (b) and (c).

```
(a)    UDRUDRUDRUDRUDRU
(b)    UDRUDRUDRUDRUDR
(c)     DRUDRUDRUDRUDRU
```

Performing a search for the last 12 pitch directions of (a) will match (c). Searching for a shorter version of (c) is currently handled by padding the query with all endings possible to make the query *L* pitch directions long.

The technique of padding the query will fail if the query is, for example, the last five pitch directions of (a): RUDRU, which is referred to here as (d). This is because there are no *n*-grams which start with (d), the window was only slid up to the end of the contour when added. To be sure of locating (d), one must pad both the beginning and the end of the query. That is, first with no padding at the beginning, then with one pitch direction of padding and six at the end, and so on. This results in a lot of padding, about $(3^{L-q})(L - q)$ extra pitch contours, where *q* is the length of the query.

This situation only arises when trying locate a contour within the last *n*-gram of a contour track. Given the small number of track endings in a database, compared with the amount of remaining *n*-grams, it does not make sense to pad short queries in this way. Doing this would reduce the overall accuracy of the database for a minimal increase in recall.

An alternative solution to the problem is to continue sliding the window past the end of the contour being added, padding to length *L* before adding. This process is considered *de rigeur* in many similar systems but they do not have the requirement that all contours in the database must be *L* pitch directions long, so they have no need of padding. If this alternative solution is taken too far, i.e. adding to the last pitch direction, 1/3 of all possible pitch contours will be associated with each track. This clearly renders the database next to useless.

Choosing an atomic length that is similar to the common query size is the best way of dealing with this issue. In some cases, the self-similarity of some contours may help. For example, the last six pitch directions of (a) will match (b) when padded in the normal way.

There is also a high chance that the contour sequence is repeated elsewhere in the target contour track. Given the small number of queries that are affected by this problem, neither of the above solutions has been pursued further.

3.5.4 Handling long queries

Queries longer than the atomic length are more troublesome than ones which are shorter. Two approaches are examined here: the first slides a window over the query; the second uses a principle from approximate matching algorithms and splits the query. Both methods produce a set of contours that may be used for a subsequent index search.

3.5.4.1 Using a sliding window

The approach first considered was to use the same sliding window technique as used when adding contours to the database (see Section 3.4). Each sub-contour is then expanded in line with the distance limit. The set of expanded sub-queries is collated in to one set. The lowest distance is used where a sub-contour occurs in more than one set.

The worst case expansion for each sub-contour is defined above, so the worst case overall will be dependent on the number of contours, which is given by q - (L - 1), where q is the length of the query. So, $n = expansion\_per\_ngram\ (q - (L - 1))$.

3.5.4.2 Splitting the query

While the sliding window method is suitable for queries which are not much greater than the atomic length, it is wholly inappropriate for very long queries, such as an entire MIDI file. A more intelligent way of splitting the query is to use a principle identified by Wu and Manber and discussed by Navarro [Navarro98]. The idea is based on the fact that for a search allowing one error in the query, the error must be either in the first half of the query or the second. An exact search can be performed for both halves of the query and the results combined. As at least one half of the query is correct, the search results are guaranteed to contain a match. This is generalised for any given number of errors, d, by splitting the query into d+1 pieces. This is the second method of query expansion that allows for errors in the query.

The use of *n*-grams has implications for this method. If the split produces queries that are smaller than the atomic length then they must be padded out by trying all permutations of pitch directions that may end each piece. For example, given a query that is 20 pitch directions long, and assuming one error, the query will be split into two pieces, each of 10 pitch directions. If the atomic length is 11 then each half becomes three queries, the original with U, D and R appended (as shown in Figure 3.6) resulting in six queries in total.

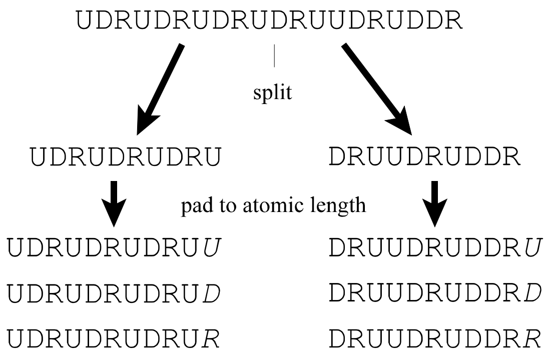


Figure 3.6 - Splitting a long query in an *n*-gram database

This example still shows an improvement on the previous approach, which would produce ten sub-contours from sliding the window before each sub-contour is expanded to allow for any errors. There is a point beyond which splitting the query will not be cost effective. If the example above used a query of only 12 pitch directions, there would be two split queries of 6 pitch directions. Completing all possible endings would result in 486 contours ($3^{(11 - 6)}$ padding per part and two parts).

As well as not allowing these split queries to be smaller than the atomic length, the use of *n*-grams requires that they must not be larger than the atomic length. Given a query of 100 pitch directions, and assuming one error, the ideal split would be into two queries which are both 50 pitch directions long. Unfortunately, this can not then be used to search the database. The best approximation to the splitting method is to segment the original query into atomic length sized chunks, ensuring that there are at least *d*+1 pieces. A simplistic implementation would move along the query making such chunks and fill in the endings for any remaining query. This is very expensive if the remainder is small, leaving a lot to pad. The number of extra contours produced by padding is 3^p, where p is the number of pitch directions to pad. It is better to take slightly undersized chunks from the query and fill in the endings. Distributing the padding in this way results in as little as $3p$ being added to the expanded query. A smaller set is preferred as this more accurately represents the query, so the search results will also be improved. The time taken to perform the index lookup will also be reduced, as there are fewer sub-contours to search for. The pseudo code for the splitting function is shown in Algorithm 1.

```
let min_ngrams = ceiling (query_length / atomic_length)
let min_pieces = distance_limit + 1
let num_pieces = max (min_ngrams, min_pieces)
let avg_size = query_length / num_pieces
let remainder = query_length mod num_pieces

while (query_length > 0)
begin
    if (remainder > 0)
        let piece = first (avg_size + 1) characters of query
        remainder = remainder - 1
    else
        let piece = first avg_size pitch directions of query
    endif

    if (piece is the last piece)
        pad the beginning of piece to be atomic_length characters
    else
        pad the end of piece to be atomic_length characters
    endif

    add the padded contours to the expanded set
    remove piece from the beginning of query
end
```

Algorithm 1 - Pseudo code for the query splitting algorithm

Note that the last piece has the beginning padded, as opposed to the end as described previously. This is required as a consequence of the way contours are added to the database which means that the last L - 1 pitch directions of a contour can not be located (see Section 3.5.3). Recall that this is because short queries are padded for all possible endings and there is nothing for this padding to match on.

Consider what happens when short pieces have their endings padded and the query consists of the end of a contour in the database (or a complete contour from the database). If the last piece of the split query has fewer than *L* pitch directions, it is not guaranteed to have anything to match against. This is illustrated in Figure 3.7, which shows a query of 31 pitch directions against the last part of a contour in the database, where the atomic length is 14 and one error is allowed in the query. The last piece of the split query (b) is 10 pitch directions, padding will try and match past the end of (a). The solution proves to be a simple one, add padding at the beginning of the last part. There must be this special case handling, always padding in this way will mean that the beginning of contours will no longer match.

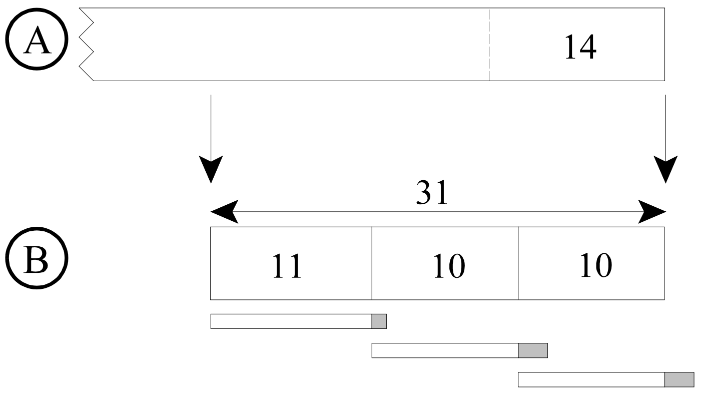


Figure 3.7 - A diagram illustrating the effect padding a split query (b) when searching for the last part of a contour in the database (a).

There are many scenarios where the queries are not likely to benefit from this alternative processing. For example, retrieval systems that process user queries are unlikely to have many instances of people searching for the last notes of a piece of music. Conversely, there are scenarios where this may be more important. One example is where a complete contour added to the database is the subject of the search (which is conceivable in the case of navigation).

One could argue that one sub-contour from a long query not being located in the index is not likely to be detrimental to the search as a whole. There are cases where the last piece is a substantial part of the query, such as the query shown in Figure 3.7. This amounts to one third of the query (81 sub-contours when it is padded) which might not match against the desired contour in the database and could be found in ‘undesirable’ contours. It is this last point, that inappropriate files might match better purely because of the padding method used, which means that the padding issue must be addressed. Omitting the last piece from the search is to be preferred above padding from the end. Perhaps a more persuasive argument for handling this case is that the processing overhead is minimal and will improve the quality of the results of a search.

3.5.4.3 Estimating the size of the expanded query set

Splitting a query is not as processor intensive as the tree-fuzzy query expansion. As such, the need to estimate the size of the expanded query set is reduced. It might well be easier simply to perform the expansion than to develop an algorithm to determine the size of the resulting set. The estimating algorithm (Algorithm 2) duplicates the initialisation of variables similar to those used in Algorithm 1 but replaces the piece handling with three simple calculations. The result, *n*, will be close to the actual number of sub-contours produced by the splitting algorithm in most cases, but it is still an estimate as it assumes that all sub-contours will be unique.

```
let min_ngrams = ceiling (query_length / atomic_length)
let min_pieces = distance_limit + 1
let num_pieces = max (min_ngrams, min_pieces)
let avg_size = query_length / num_pieces
let remainder = query_length mod num_pieces

let n_per_peice_with_r = $3^{(atomic\_length - (avg\_size + 1))}$

let n_per_peice_without_r = $3^{(atomic\_length - avg\_size)}$

let n = (remainder * n_per_peice_with_r) +
            ( (num_pieces - remainder) * n_per_peice_with_r )
```

Algorithm 2 - Pseudo code for the estimating the number of sub-contours when splitting the query

3.5.4.4 Concepts of error tolerance

The combination of handling of long queries and query expansion results in two concepts of error tolerance. The first is that introduced by tree-fuzzy query expansion. This operates on each atomic length window on a query that is processed by using a sliding window. This can be thought of as a local error tolerance, or the *local distance limit*. For example, on a query of 13 pitch directions, and assuming an atomic length of 12, a local distance limit of 1 will allow for 2 errors in total.

The second concept is the number of errors to allow in total, the *global distance limit*. The query splitting algorithm uses this limit for deciding on the minimum number of pieces a query must be split into.

3.5.4.5 Selecting the best expansion method

For the fastest index search, the number of sub-contours in the expanded query set should be kept to a minimum. The most appropriate expansion method can be determined by calculating the size of the sets each method would produce. Ideally, the calculations would return precise figures but this is not crucial. Anecdotal evidence has indicated that a rough approximation is good enough to select an appropriate method. Anecdotal evidence also indicated that, while this strategy results in the fastest search, using a slightly larger expanded query set could help improve the accuracy of the search.

3.5.5 Multistage queries

The use of *n*-grams has the side effect that a longer query, whilst returning a more relevant top choice, will usually return more matches no matter how the query is expanded. This is similar to the behaviour exhibited by some Internet search engines, such as “AltaVista”. The more information a search engine is given, the more relevant the top suggestion is likely to be, although many millions of documents may partially match. This is due to the fact that no checking is done to ensure that the whole query is actually contained in the matched files, or indeed that they occur next to each other (that they are temporally aligned).

It is reasonable for a user to expect that a longer query would result in fewer matches being found. This is not the case because, as described above, a larger expanded query set will find more matches in the index. The user's expectations can be met to a limited extent by sanity checking the results of the search. Firstly, there will be a minimum number of sub-contours in the query that must present in any result for it to have chance of being a match. For example, a long query of length 31 might be split into three pieces. This means that each result must indicate that at least three contours match to be valid. The calculation of the minimum number of hits is dependent on the method of expansion. If the sliding window method was employed: *min_hits* = (*query_size* - *atomic_length*) + 1. Splitting the query require the use of: *min_hits* = ceiling(*query_size* / *atomic_length*). This works, regardless of the error tolerances used, although it not likely to filter out many results when higher tolerances are used, as more sub-contours are present in the expanded query set and so the hits per result will be higher.

The second method of sanity checking, which can be used in conjunction with the first, requires that the index be searched with a local distance limit of zero. It is based on the principle that, for an exact match, all sub-contours in the expanded query set must be present in any valid result. This means performing an AND search for the query set. This technique can be enhanced to cater for a global error tolerance by ensuring that no more than *global_limit* sub-contours are missing from each result.

An unfortunate side effect of expanding long queries is that, while the index can determine that a file contains at least some of the query, it alone can not confirm a match for the whole query. There is nothing to guarantee that the parts of the query that matches are temporally aligned. As such, the use of *n*-grams requires long queries to be validated, in theory at least. Downie [Downie99] claims that performing this validation did not noticeably improve the accuracy of his results.

3.6 Matching using secondary contours

Pitch contours are designed as an abstraction of the exact pitch to allow for errors in the input. This is useful for both hummed queries and queries produced by using feature extraction engines. As well as allowing low quality queries to locate music requested by the user, the pitch contour can also produce a large number of false matches. The pitch contour for two pieces of music can be identical, even though they might be very different to listen to. The example score in figure 3.8 shows this, where the two bars have the same pitch contour of ‘UDU’.

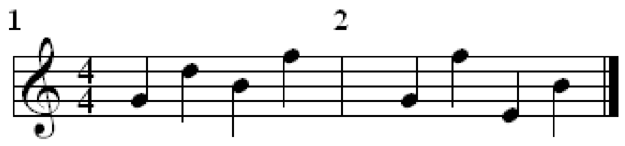


Figure 3.8 - Two different bars with the same primary contour

While pitch errors are likely to accumulate over time with many query sources, e.g. humming, more information can be deduced from notes which are close together. One improvement would be to classify intervals out of five possible types: up, up a lot, repeat, down and down a lot. The classification for up, down and repeat is the same as the one discussed previously. The distinction between ‘up’ and ‘up a lot’ could depend on a threshold on interval size, but a more reliable approach is to compare a note with a pitch previously established in the contour; e.g. if the current note is of higher pitch than the note before the last one, then it is classified as being ‘up a lot’. It should be noted that this only adds more information when the pitch direction changes. For example, in a sequence of ascending notes all but the first pitch direction would be ‘up a lot’. The five pitch classes can be encoded as a single character representation of the direction types: u, U, r, d and D for up, up a lot, repeat, down and down a lot respectively. Using this representation, the first bar of the example (Figure 3.8) would be ‘udU’, while the second bar would be ‘uDu’, showing that the two bars are indeed different.

An alternative approach has been adopted which has a similar effect but retains the existing representation. A secondary contour is created where each symbol represents the relationship between the current note and the ‘note before last’; e.g. in the example above, this secondary pitch contour would be ‘UU’ for the first bar and ‘DD’ for the second bar. This concept of secondary contours could be generalised to *n*-level contours, where the pitch of the current note is compared with the *n*th previous note.

Using the existing representation allows the same database structure to be used. A second database that contains the secondary contours is stored alongside the primary contour database. The lookup procedure is carried out for each database and the intersection of both sets of results are taken, as both the primary and secondary contours must be present in any matching file. This noticeably reduces the list of matches while maintaining a quick response time.

This initial query only identifies files which contain both the primary and secondary contours. Further improvements are made by checking that the contours are aligned: ensuring they happen at the same time in the file. This uses the cached contour files that were created when the MIDI file was added to the database.

3.6.1 Comparison with five pitch classes

To compare the expressiveness of primary/secondary contours with that of a single five pitch class representation, a list of all theoretically possible combinations of primary and secondary contours for three consecutive notes was produced. An example was then found for each combination, where one existed. This is shown in Figure 3.9, where it can be seen that only 13 combinations out of the 27 possible are valid.

More importantly, there are only 13 ways of arranging three consecutive notes. This can be proved by considering the number of positions that are available to each note, with respect to the previous notes. The product of the number of positions can be taken as the number of arrangements. The first note is used as a reference, as there is no previous note, so can be considered as fixed and only having one position. The second note can be either: higher than the first, equal to the first or lower than the first. This gives three possible positions. The third note can be either: higher than both previous notes, equal to the first, equal to the second, between the first and the second or lower than both the previous notes. This gives five possible positions for the third note. This would suggest that the total number of arrangements is 15 (1 x 3 x 5) but this is not so, as there are two special cases. If the first two notes are the same, then there can be no “between the first and the second”. There are also duplications when the first two notes are of equal pitch, as “equal to the first” and “equal to the second” will produce the same arrangement, making one of them redundant. These two cases reduce the total number of arrangements to 13.

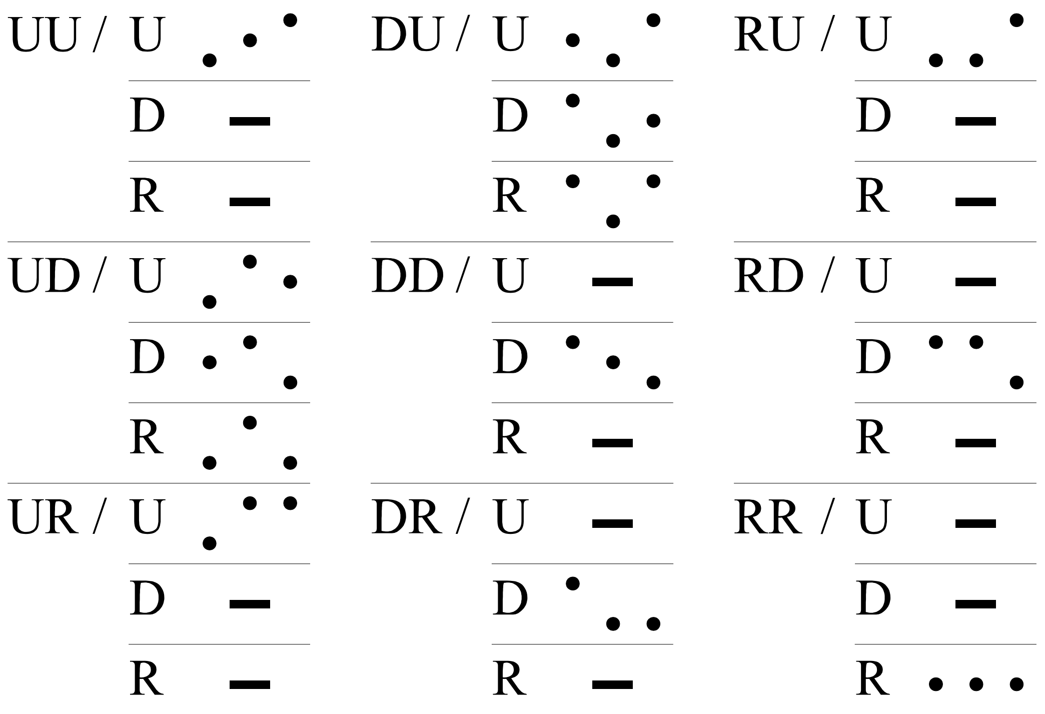


Figure 3.9 - All possible primary / secondary contours for three notes, with examples of each. A dash indicates that no example could be found

The secondary contour is constrained by the primary. For example, for three notes which increase in pitch, the primary contour is ‘UU’. If the third note is higher than the second, and the second higher than the first, the third note must be higher than the first. It can not be lower, or even the same pitch as, the first note so the secondary contour can only be ‘U’.

It can be seen that the five pitch class representation has no way of denoting a repeating pattern as shown in the secondary contour, so ‘UD/U’ (primary contour / secondary contour) and ‘UD/R’ both translate to ‘ud’. It is clear that converting a contour from two three-class contours into one five-class contour will lose information which can not be regained. The conclusion must be that using two separate contours is more expressive in this respect than a single contour with more pitch classes. With two extra representations for a repeat, the use of a secondary contour gives the equivalent of seven pitch classes, although having three types of repetition might not be as useful as dividing the pitch scale into seven segments. It does, however, incorporate an element of temporal information, e.g. showing that a pitch is repeated every other note.

Table 3 - The translation of primary/secondary contour combinations to a representation with five pitch classes

Primary	Secondary	5 Class
UU	U	uU
UD	U	ud
UD	D	uD
UD	R	ud
UR	U	ur
DU	U	dU
DU	D	du
DU	R	du
DD	D	dD
DR	D	dr
RU	U	rU
RD	D	rD
RR	R	rr

Table 3 also shows that the secondary information is only useful when there is a direction change in the primary pitch contour, i.e. ‘UD’ or ‘DU’, as the secondary contour may be inferred in all other cases. For example, the primary pitch contour ‘UU’ always has a secondary pitch contour of ‘U’.

3.6.2 Comparison with p-structures

The Query by Pitch Dynamics [Beeferman97] retrieval system uses p-structures to maintain the relationships between all notes except the precise interval. For example, it can directly show that the first and last notes are the same (or that they are different). The same is not true of primary contours, as they only specify the relationship between the current note and the previous one. There are cases where more information is implied, e.g. the contour of ‘UU’ implies three distinct notes, but the worst case will not contain any implicit relationships (e.g. ‘UD’). A similar argument applies to secondary contours as they only guarantee to specify the relationship between the current note and the previous two.

From the above, it can be deduced that *n*-ary contours are as powerful as p-structures only when *n* is no less than the length of the query minus one. This is impractical for queries of more than a few notes but would be more flexible than p-structures, as the level of accuracy required of a contour could be defined on a per query basis.

3.7 Comparison of accuracy

The system of music retrieval that is described in this chapter has a number of variables that will affect the accuracy of a search. The most important ones are the distance limit, the atomic length and whether secondary contours are used, or not. There is an additional implied variable: the number of files in the database.

This section evaluates the accuracy of the database and answers questions such as: are secondary contours useful? What happens to performance as errors in the query are accounted for? What happens to performance as the atomic length is varied? Databases using atomic lengths of 9, 12, 13 and 14 were compiled for the evaluation.

The quality of matches was evaluated by performing searches for pieces of music which were known to exist in the database. The rank of the target piece (if found) was noted, along with the total number of matches found. Database queries were simulated by extracting segments from MIDI files in the indexed collection. The set of queries was automatically generated and consisted of 500 MIDI files, each being 10 seconds long. The queries were required to be at least 14 pitch directions long. This is a reasonable length for a query of that duration, given the earlier observation that the collection averages two pitch directions a second. The restriction is, in part, also related to the fact that the largest atomic length considered here is 14. Evaluation when allowing for short queries would be misleading as it becomes difficult to determine if the results of the search are poor due to the query parameters or the query itself. It should be noted that these randomly generated queries might not necessarily reflect human generated ones but is the easiest way of obtaining a statistically significant number of queries by which to evaluate the system.

Three types of measurements were considered for evaluating the quality of the results. Recall is not discussed here because in all tests it was 100% and so is not a useful discriminator. Given the knowledge that the target of the query is always found, where it appears in the ranking is a more informative measurement. Also, is 100% recall achieved by the number of matches being so large that the target is likely to be included by chance, i.e. is the whole database returned each time? The *mean rank* and *mean set size* metrics are aimed at measuring this. While the average quality of searches is obviously an important factor, it is not necessarily a good indicator of the performance as perceived by a user. This is because a few extreme values can have a significant influence on the mean value. To counter this effect, the rank by which the majority of targets are found is recorded. A majority is considered here to be 85%, or more, of queries. This allows statements of the form “the majority of queries rank their targets in the top four matches”. In a ranking system, this is arguably more relevant than the number of files returned.

The usefulness of secondary contours was evaluated by considering the effect they had on a database with an atomic length of 12. Figure 3.10 shows that there is a linear relationship between the size of the database and the ranking of targets. In addition to this, it clearly shows that secondary contours are highly effective in improving the ranking of targets. Note also the difference between the average performance and that perceived by a user. There is a similar linear relationship between the size of the database and the number of matches returned. Again, secondary contours significantly improve the results, halving the number of matching files found from an average of 396 to just 183 (for a database with 7,600 files). Given the clear advantage that using secondary contours has over not using them, the rest of this chapter only considers the results of tests performed when using them.

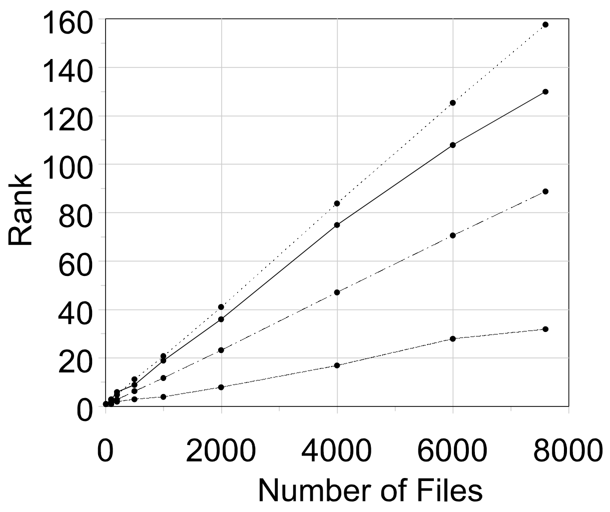


Figure 3.10 - A graph evaluating the use of secondary contours. From top to bottom: mean rank with primary, majority rank with primary, mean rank using secondary, majority rank using secondary

Figure 3.11 shows the effect that allowing for errors in the query has on the ranking of targets. It gives this information for two databases, one with 100 files and one with 1000 files. The ranking of targets when performing an exact search (*d* = 0) on a 100-file database is good. In this case, 86% of queries placed the target as the top match and over 91% were in the top two. The other extreme is the performance on the larger database and allowing for two errors in the query. Here, only 50% of queries will return the appropriate piece in the top six matches.

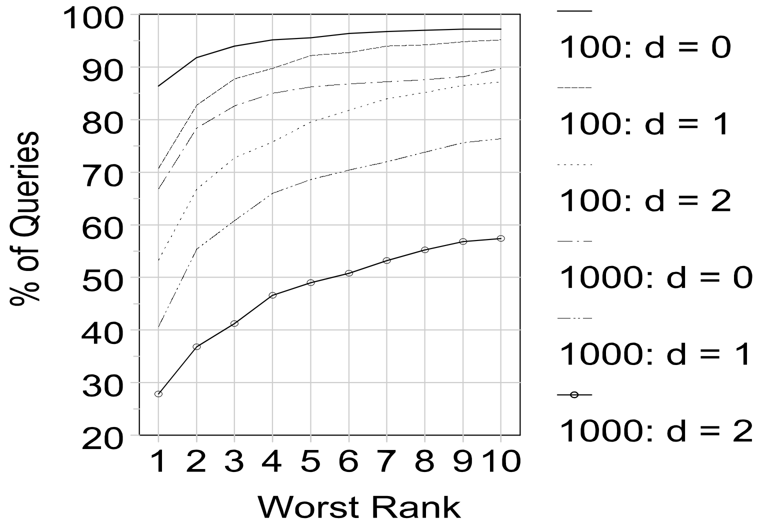


Figure 3.11 - A graph showing the effect of allowing for errors in the query

The evaluation proved that the quality of the results can be improved by increasing the atomic length, *L*. The average rank of targets is markedly better when *L* = 13 compared to *L* = 12 (see Figure 3.12). The advantage gained by increasing L from 13 to 14 is not so great. Smaller atomic lengths become increasingly ineffective. The average ranking of targets in a database of 7,600 files that has an atomic length of 9 is 313. This is equivalent to approximately 4% of the database, making it ineffective for all but the smallest collections of files. A similar relationship was observed for the ranking of the majority of queries (Figure 3.13), although the relationship is not so uniform. Interestingly, this shows the ‘majority’ rank to be worse than the average for *L* = 9, being 445 for the largest database.

The number of matches found follows a similar pattern to the other two metrics. Figure 3.14 shows the number of matches found to increase linearly with the number of files in the database. The gradient of this relationship is lower as the atomic length is increased. It shows again that an atomic length of 9 is too small, returning 843 matches (on average) for the largest database.

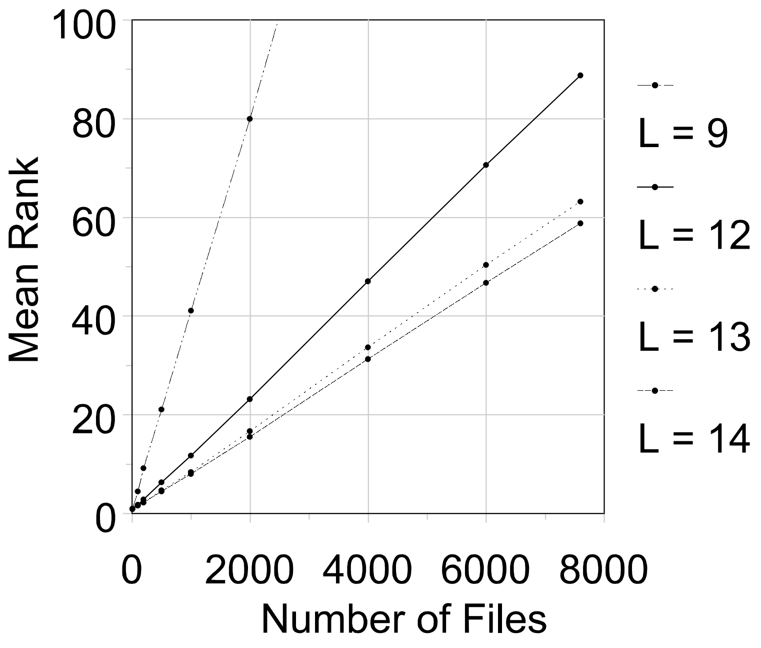


Figure 3.12 - A graph showing the mean rank for the different atomic lengths and database sizes

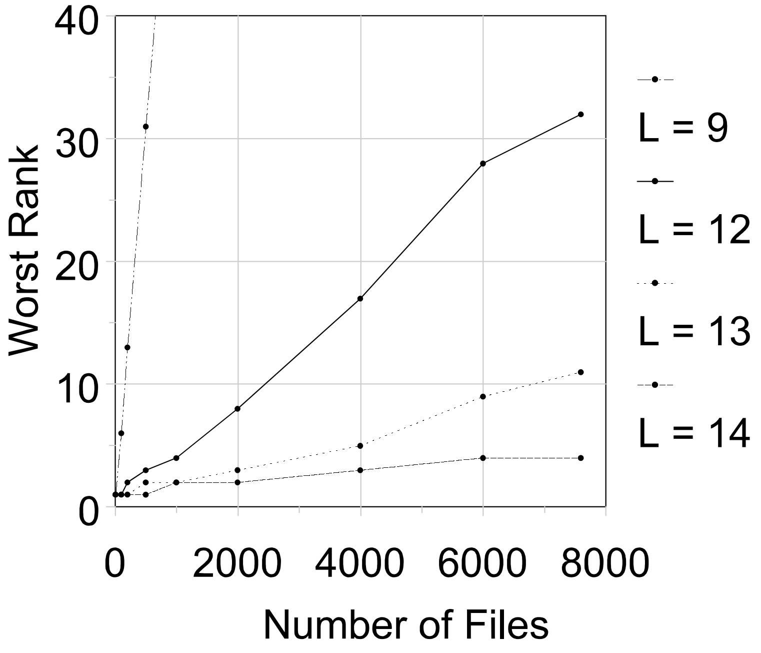


Figure 3.13 - A graph showing the rank of the majority of queries the different atomic lengths and database sizes

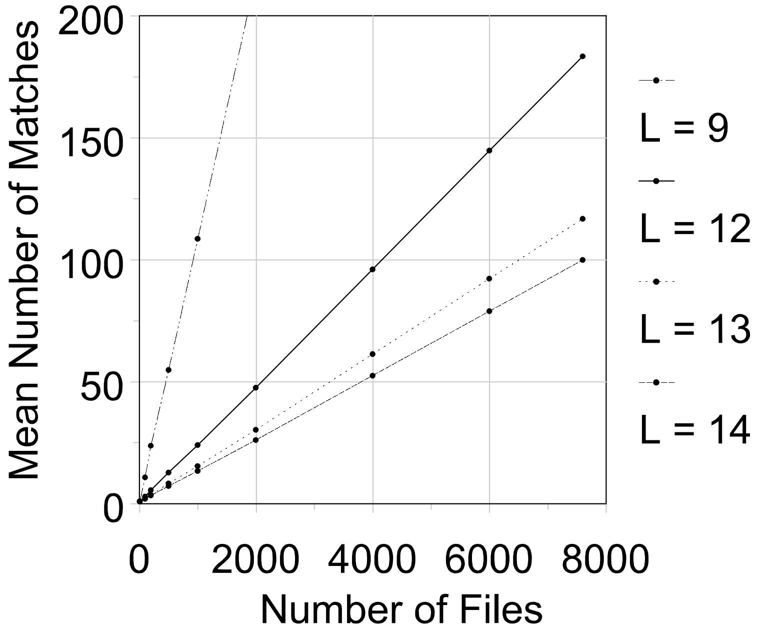


Figure 3.14 - A graph showing the mean number of matches found for the different atomic lengths and database sizes

In addition to the simulation of an operational database, the nature of the custom database, an inverted index of the entire set of MIDI files, made it possible to determine the performance for optimal (atomic length) queries directly.

Each possible sub-contour in the database was examined and the number of files which contained it was recorded. The result was a list of the number of sub-contours (essentially atomic length queries) and the number of files in which they occur. More queries matching in fewer files suggests that there will be fewer false matches.

The results are plotted in Figure 3.15 and Figure 3.16. The first graph shows that increasing the atomic length *L* does improve the resolution of the database. That is, more sub-contours are contained in only one file. An efficient database is one whose lookup table does not have much unused space. To get an idea of how the efficiency of the database is affected, in Figure 3.16 the same results are plotted against the percentage coverage of the database. This shows that over 25% of the possible contours with 13 pitch directions match just one file. Observe that the efficiency is decreased for greater *L*.

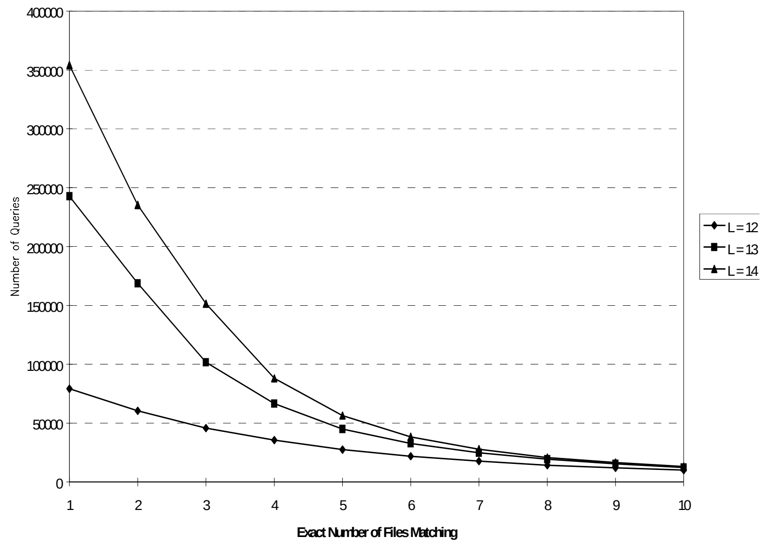


Figure 3.15 - The number of contours *vs.* exact number of matching files

The average number of files returned for each database is shown in Table 4. The decreasing average, together with the first graph, suggests that increasing *L* does improve performance, although efficiency is subsequently decreased.

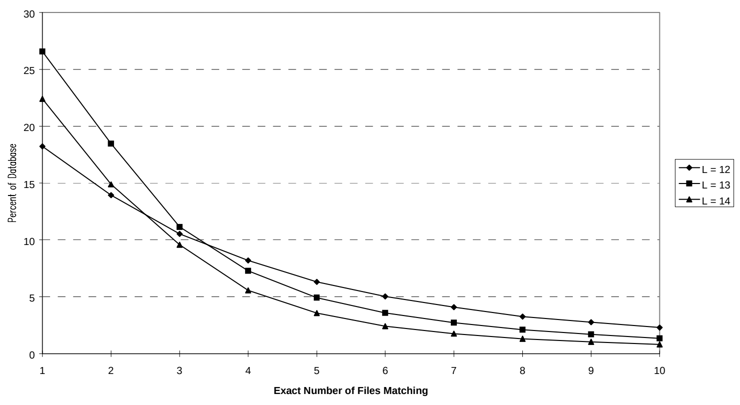


Figure 3.16 - Percentage coverage *vs.* exact number of matching files.

Table 4 - Average number of files returned for different atomic lengths

L	**Average**
12	16.97
13	9.71
14	5.99

3.8 Comparison of speed

The database of 7,600 files was indexed using the custom database structure in about three and a half hours. Files are added to the database at an average rate of 35 per minute, or one every two seconds. The primary concern is the speed of lookup as, for most applications, the index does not need to be modified interactively.

The time taken for an atomic length query was measured by generating random primary and secondary contours and performing an exact search on those contours. This was repeated as many times as possible in ten minutes. An average execution time could then be calculated. This was done for each database type. From Table 5, it can be seen that the custom database is the fastest but the text retrieval engine is less than half a millisecond slower. The standard database is clearly much slower.

Table 5 - The average time taken for an atomic length query

Database type	**Time per lookup**
Custom format	1.75 ms
Onix	2.16 ms
Standard database	40.16 ms

The time taken for the accuracy tests to execute was measured for the custom database. This information was then used to calculate the total time taken for processing a query, including extracting a pitch contour from the query file. On average, queries of the ten second clips took 70.8 ms to execute. While observing the progress of the tests, it was noticed that the majority of queries were executed much more quickly than this, perhaps three or four times quicker. This was offset by a few searches that took several seconds to complete. A busy link server could take advantage of this behaviour by increasing throughput by identifying and ignoring queries that are known to be slow. This would be particularly beneficial during peak usage times.

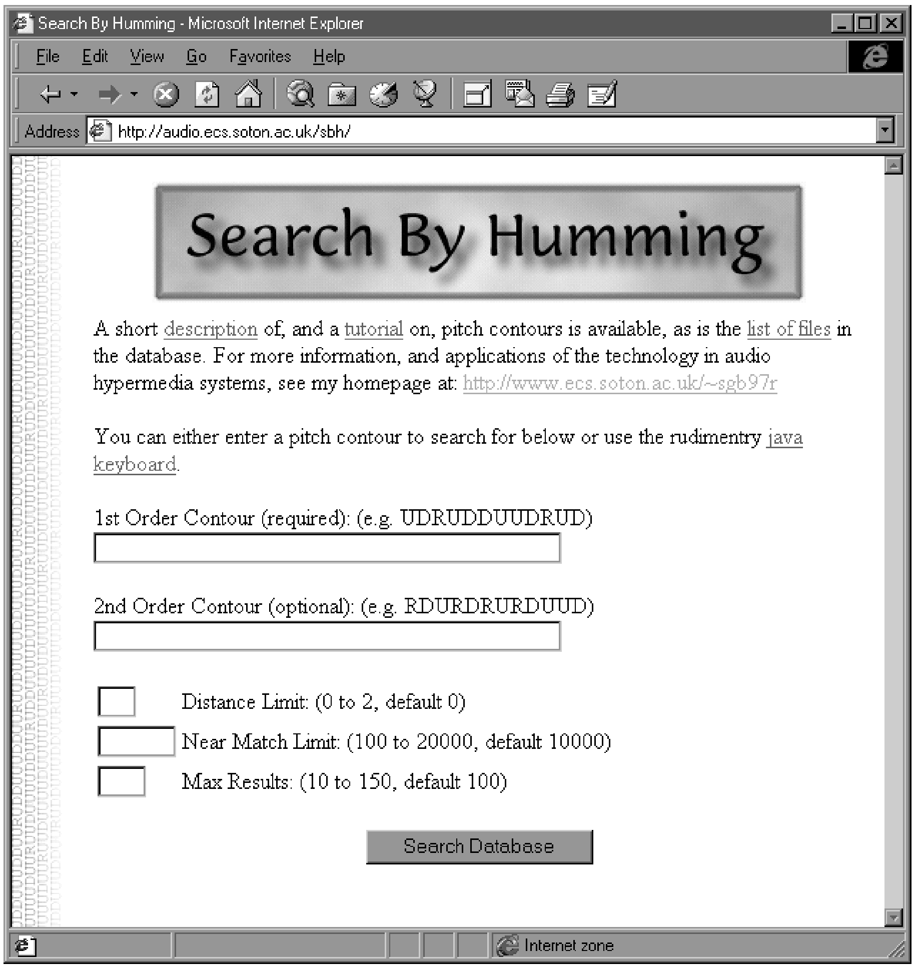
3.9 A web-based music retrieval system

A Common Gateway Interface (CGI) was developed for the retrieval system. The CGI is a method of dynamically creating web pages. The user can type a URL into a web browser which executes a program or script on the web server that generates the appropriate HTML document. Arguments to the executed code can be embedded in the URL, such as the search terms for an Internet search engine. It is common for the user to enter these arguments via a HTML form.

An index for small collection of MIDI files was installed on a web server. The index can be searched by primary and secondary pitch contours, entered on the main page (Figure 3.17). From the results page (Figure 3.18), the user can play a matching MIDI file or identify where in the song the match, or matches, occur.

Although psychological research has suggested that pitch contours play a strong part in how melodies are remembered, or at least compared, it is not an easy representation for the user to create directly. A small number of people were asked to write down the pitch contour for Happy Birthday (i.e. from memory to paper) and only a few transcribed it correctly. This is despite the fact that most people hummed the tune correctly. It is for this reason that an alternative interface was sought.

The Java piano applet (Figure 3.19) allows the user to enter a tune using a familiar interface (even to non-musicians). The pitch contour for the entered tune is generated automatically on initiating a search. In practice, this creates a URL of the same form as that created by the web form and so displays the same results as for the form-based interface.


Search By Humming - Microsoft Internet Explorer
File Edit View Go Favorites Help
Address http://audio.ecs.soton.ac.uk/sbh/
Search By Humming

A short description of, and a tutorial on, pitch contours is available, as is the list of files in the database. For more information, and applications of the technology in audio hypermedia systems, see my homepage at: http://www.ecs.soton.ac.uk/~sgb97r

You can either enter a pitch contour to search for below or use the rudimentry java keyboard.

1st Order Contour (required): (e.g. UDRUDDUUDRUD)

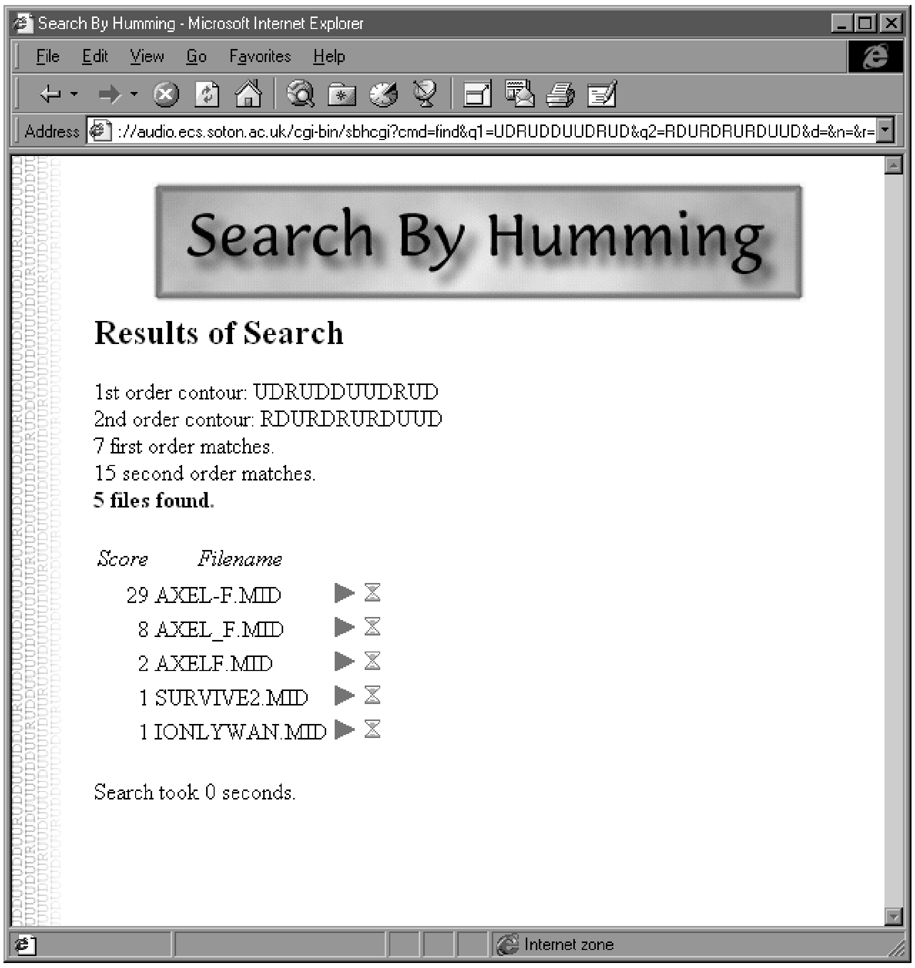
2nd Order Contour (optional): (e.g. RDURDRURDUUD)

Distance Limit: (0 to 2, default 0)

Near Match Limit: (100 to 20000, default 10000)

Max Results: (10 to 150, default 100)

Figure 3.17 - The search form of the web-based interface


Search By Humming - Microsoft Internet Explorer
File Edit View Go Favorites Help
Address ://audio.ecs.soton.ac.uk/cgi-bin/sbhcgi?cmd=find&q1=UDRUDDUUDRUD&q2=RDURDRURDUUD&d=&n=&r=
Search By Humming
Internet zone

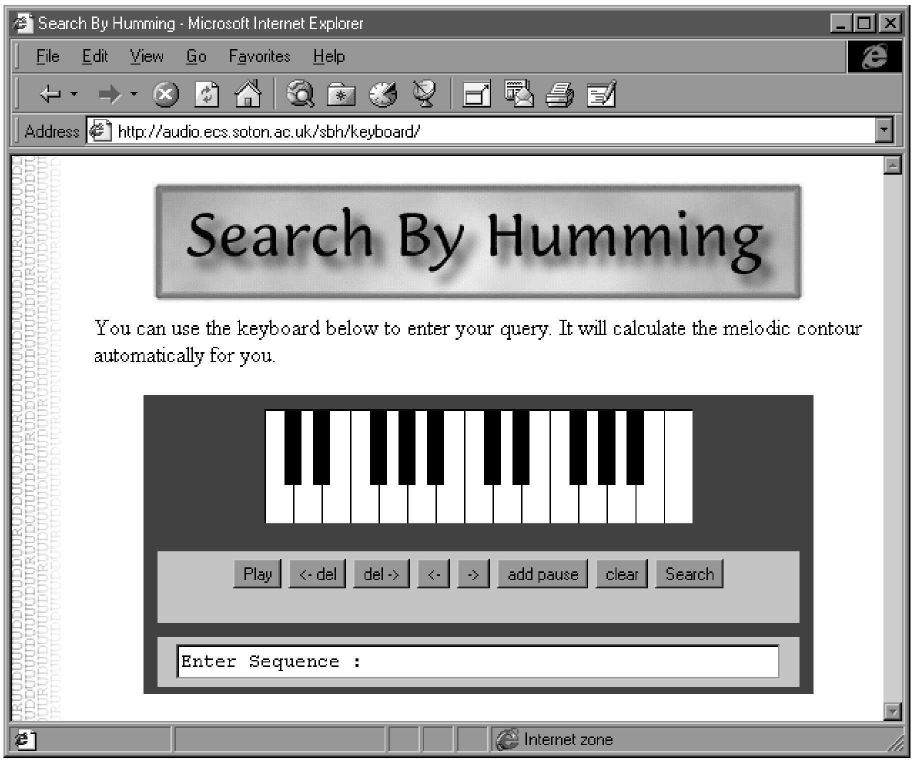
Results of Search

1st order contour: UDRUDDUUDRUD
2nd order contour: RDURDRURDUUD
7 first order matches.
15 second order matches.
5 files found.

Score	*Filename*
29	AXEL-F.MID
8	AXEL_F.MID
2	AXELF.MID
1	SURVIVE2.MID
1	IONLYWAN.MID

Search took 0 seconds.

Figure 3.18 - Search results as presented by the web-based interface


Search By Humming - Microsoft Internet Explorer
File Edit View Go Favorites Help
Address http://audio.ecs.soton.ac.uk/sbh/keyboard/
Search By Humming
Play
<- del
del ->
<-
->
add pause
clear
Search
Internet zone

You can use the keyboard below to enter your query. It will calculate the melodic contour automatically for you.

Enter Sequence :

Figure 3.19 - The Java piano applet

The web-based retrieval engine is functionally equivalent to a remote lookup service. Any client program that is connected to the Internet can use the web interface to perform content based retrieval. This model is similar to the approach taken by agent systems. As part of the Agent Fest, a combined effort to write agents for SoFAR (an agent framework), a Java-based agent performed melodic pitch contour lookup by using the above mechanism. An agent that extracted text contained in MIDI files was developed in a similar manner, using a CGI / agent combination.

The retrieval software developed for this thesis has been used more directly by Matt Swoboda. Matt developed an application that watches what a musician is playing, using a MIDI instrument, and identifies what is being played at any given moment. Regarding the accuracy of search results, he says:

> “The system works very well for a user who plays badly and expects to find their desired result somewhere in the rankings but perhaps not at the top. It performs less well for a perfectly played performance.”

This is to be expected, both due to the searches allowing for one error and because pitch contours assume that the exact pitch can not be relied upon. He notes that there are some contours (e.g. all R's or all U's) that are reported as matching in many files. He proposes that this is a shortcoming of pitch contours. This may well be the case for all U's but is unlikely to hold for a sequence of repeated notes, no amount of pitch classes will represent this differently. In this case, it is more likely that repeating notes are so common that they should be removed from the query. A large list of matches is probably less helpful than returning no matching documents.

3.10 Augmented searches

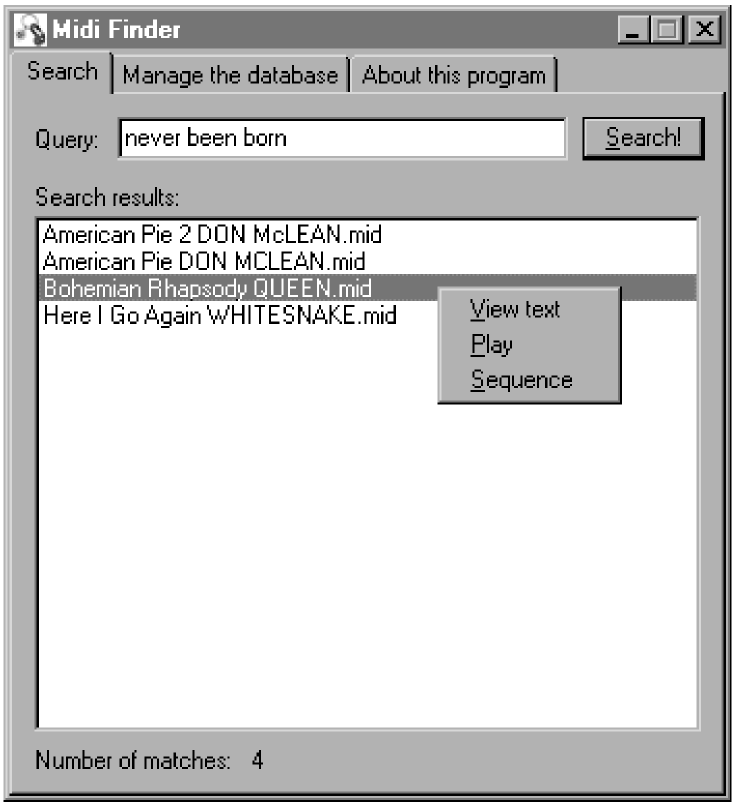
It may be the case that a user knows the artist or title of a piece of music they are searching for. This information could be used to narrow the search if metadata is available, such as if MIDI files had a standard method for storing meta information, e.g. title and artist. The MIDI file format has the provision for naming each track. How this is used is dependent on the person sequencing the file but common uses are to label each part (drums, vocals, lead) or to label each instrument (piano track, violins, brass section) or sometimes to note the artist and title of the song, along with the name of the person who sequenced the file.

The fact that there is no convention for storing this information makes it impossible to extract just this information reliably. Instead, all the text contained in the file must be used. A text retrieval database can then be employed to index the text. It must be noted that not all files contain the relevant information, so the candidate files presented by a melodic search should not be culled just because they do not feature in the results of a title search. The text search can only reliably be used to increase the matching relevance score of the files which contain that text.

There is a class of MIDI file in which the lyrics that accompany the music are embedded as lyric events. Each lyric event has timing information, so that a software karaoke player can display each lyric at the appropriate time. These lyrics may be extracted and indexed using the same method as described above, so being used to augment a search. There are also applications for content based navigation, as the lyrics for a particular segment of music may be extracted as content to be used to resolve a link.

The concept of extracting and indexing text contained in MIDI files was tested by developing a piece of software that did just that. It used the same text indexing library as given in Section 3.4.3. A collection of 870 karaoke files was added to the database.

A screen shot of the software is shown in Figure 3.20, and the test has shown that the idea is a sound one. Although no formal testing was carried out, it is possible to make some comments on the development. Indexing is fast, informally measured at over eight files per second, and retrieval is even faster. A lot of this will be down to the performance of the text search engine used, which is not the subject of this thesis and so testing this formally was thus deemed out of scope.



Query: never been born

Search results:

American Pie 2 DON McLEAN.mid
American Pie DON MCLEAN.mid
Bohemian Rhapsody QUEEN.mid
Here I Go Again WHITESNAKE.mid

Number of matches: 4

Figure 3.20 - Screen shot of the MIDI text indexing and retrieval tool

3.11 Conclusions

This chapter has described a fast retrieval method which uses contour *n*-grams to index a collection of MIDI files. It allows approximate lookup of the index through the use of query expansion. Errors can be simulated in short queries while long queries may be split to avoid errors. These query management techniques may be used independently of any particular indexing method. The use of the custom database format has been shown to be preferable, although the text retrieval engine comes a close second.

The results of the index lookup must undergo several stages of verification before a concrete list of matches can be presented to the user. In practice, the final check against the source file is not usually required. The use of a new representation, the secondary contour, as an additional representation proves to be worthwhile.

The handling of long queries is currently optimised for fast index lookup but this can result in a trade off against the accuracy of an index only lookup and can actually take longer, as there are more candidate files.

As they stand, the techniques presented in this chapter are suitable to form the basis of a reasonably sized content based navigation system. Over 80% of ten second queries have been shown to return the target song in the top three from a database of 1000 complete songs. It is not reasonable to expect similar performance when substituting the song database with a collection of 1000 hypermedia anchors.

The retrieval system presented tries to make the best use of the data placed in the index. Before considering navigation of audio in depth, it is worthwhile investigating the quality of this data. There were over 30 million pitch directions in the primary contour database, averaging 120 pitch directions per minute of each MIDI track. Given that the number of songs that have ever been written is orders of magnitude greater than that used to test the database. It seems obvious that reducing the amount of data being added, while maintaining recall accuracy, is the best way of ensuring the system will scale and should improve performance for the same unfiltered database. An approach to this problem is considered in the next chapter, which uses part classification to filter out irrelevant MIDI tracks.

4 Enhancing retrieval by classifying musical parts

Classification can improve content based retrieval systems by decreasing the amount of noise in the database. Taking retrieval of music as an example, there is little point in indexing content which is not going to be the subject of subsequent searches. For example, removing all but the lead (melody) line should improve the matching ability of a query by humming system. Alternatively, a music education application might focus on one part, such as the bass line. Content based navigation can also benefit from classification by this extra information by allowing it to be used as another component of the link.

This chapter describes the investigation into the classification of the musical parts. The effect of applying the technique to a database of music is subsequently evaluated. This chapter presents an extension of the work reported in [Blackburn2000].

4.1 Introduction

Classification has uses in both temporal and content based navigation, although one could argue that the temporal case is a form of the latter: content may be extracted from the source document using the temporal information. The classification could be used as a parameter to link resolution. This enables temporal links to be authored on particular parts of the music, i.e. from 20 seconds to 30 seconds of the bass line. Similarly, content based navigation would allow an anchor to take the form of ‘lead lines that sound like this’.

The MIDI databases developed by Ghias *et al* [Ghias95], McNab *et al* [McNab96] and Blackburn and De Roure [Blackburn98] make use of the General MIDI (GM) standard, which defines MIDI channel 10 to be dedicated to drums. Assuming that all MIDI files adhere to the standard, which the majority do, rhythm tracks can be identified with a high degree of certainty and not added to the database. General MIDI can also help identify instruments, which could be relevant in some applications, but of course does not identify the role that instruments play in the piece.

In some scenarios, such as some implementations of content based navigation, the track being classified may be taken out of the context of the other tracks. This implies a requirement that music must be classified without the benefit of comparing the classification subject with the rest of the piece. Removing this requirement should make identifying the lead line relatively simple, as several studies suggest that this is likely to be the track with the highest pitch [Uitdenbogerd98]. Common sense would suggest that the bass line would have a lower average pitch than the lead.

4.2 Part types

Four musical part classes were used which were aimed at how a layperson listener is likely to identify each part. This approach avoids the use of any genre specific terminology and so is more generally applicable. The classifications used were: accompaniment; bass; drums / rhythm and lead.

The use of a classification to describe irrelevant parts was considered but it was felt that it would simply introduce noise into the classifiers, making identification more difficult. Using a separate confidence measure, allowing a ‘not known’ classification, would be more appropriate, although not all classifiers are capable of providing this. While none of the implementations in this chapter calculate such a measure, none of the algorithms here preclude adding this functionality.

4.3 Compiling the training set

Classifiers require a set of files by which to evaluate their performance. Many require the same data to implement the classifier. One hundred MIDI files were selected at random from the test collection of 7500 files, see Section 3.2. These covered a variety of genres. Certain rules may be assumed if it consisted of a single genre, e.g. popular music. The variable quality and variety of musical style makes classification that much harder, but the result should be a more generally applicable classifier. The part classification for each track in each MIDI file was marked up manually. The tracks were then extracted to individual MIDI files (so there was only one track per MIDI file). This resulted in a total of 535 classified musical parts: 152 accompaniment; 103 bass; 144 drums and 136 lead.

4.4 Musical features

Classification requires the extraction of a set of features, referred to as a feature vector, from each track. Two feature vectors can be compared by determining the Euclidean distance between them. The Euclidean distance is found by summing the squares of the difference between each element in a vector and then taking the square root. Another function evaluates the 'city block' distance. This is also referred to as the absolute distance, as it is simply the sum of the differences.

A variety of features were extracted from each training track. Features were chosen that could be represented as a single number, with the exception of number 5. There are a large number of features that could have been extracted. A selection of features which the author thought might be useful in identifying the part type were used. The selection was based on informal discussions with colleagues and friends. These were:

1. The number of notes per second
2. The number of chords per second
3. The pitch of notes (lowest, highest and the mean)
4. The number of pitch classes used (i.e. C, C#, D, E, etc)
5. Pitch classes used
6. Pitch class entropy
7. The length of notes (lowest, highest and mean)
8. Number of semitones between notes (lowest, highest and mean)
9. The polyphony of the track (lowest, highest and mean)
10. How repetitive the track is
11. The number of the MIDI channel used
12. The instrument patches used most

All of these features are normalised to a value between 0 and 1, as this improves the performance of neural networks for classification. It might also improve the effectiveness of other classification systems and can make calculation of a probability straightforward.

Repetitiveness is a number based on how many distinct *n*-grams there are in the primary pitch contour of the track. This is represented as a percentage of the number of pitch contours in the track. If there are 20 distinct contours, in a track which has 100 pitch contours in total, the repetitiveness value would be 0.8, indicating that 80% repeats. For this work, $n = 12$ was used. This is not a standard measurement of repetitiveness; there is no one standard technique available as it relies on the representation used. The use of *n*-grams throughout the database made this approach easy to implement and was the overriding factor in the decision to use it.

4.5 Approaches to classification

Five methods of classification were considered and are described in this section. They were chosen to be representative of different approaches to classification. Some techniques are purely statistical while others attempt to bring to bear some knowledge of the nature of the content. A learning algorithm is represented in the form of a neural network. A combination of all these approaches completes the selection.

4.5.1 A simplistic statistical approach

The first approach taken was to average each of the features for all of the tracks of a common part classification. The result was four feature vectors which describe the features common to each class type. A feature vector to be classified is compared with each of these class templates. The class of template that matches the closest indicates the part class.

Although not particularly scientific, this technique could be implemented quickly and provided a benchmark by which to evaluate the more complex approaches.

4.5.2 An ad-hoc classifier

The ad-hoc classifier was based on directly coding the knowledge of a musician. A discussion was held with an experienced music student to determine which of the features he felt would best help identify part types. The implemented classifier notes the features in a channel that make it more or less likely to be a particular part. A probability can then be assigned to each part and the most likely chosen.

As previously mentioned, General MIDI (GM) files assign channel 10 to drums. In addition to this, there are some GM instrument numbers that are usually used for rhythm channels (i.e. timpani, taiko drum, melodic tom, synth drum, reverse cymbal and wood block). Tracks that use channel 10 or an instrument that is usually used for percussion are assumed to be drum tracks. The probability of other tracks being drums is taken to be 70% of the repetitiveness measure.

A track is more likely to be accompaniment if it only uses notes which are no more than one octave either side of middle C and is likely to be polyphonic. Bass parts tend to use the two octaves below the accompaniment and are quite repetitive (a measure greater than 0.6). Bass lines are very likely to be monophonic and are heavily penalised if they are not. Lead parts often use the two octaves above middle C and are also likely to be monophonic. They are penalised for being repetitive.

4.5.3 K-nearest neighbour

The *k*-nearest neighbour (K-NN) classification method is, theoretically at least, the most accurate one considered here. The entire training set of feature vectors is stored, along with their part classification. Classifying a vector finds the example vector that best matches the unclassified one and returns the classification of that example.

Due to the relatively small size of the training set (hundreds instead of thousands) it was feasible to keep the entire set in memory. Searching against the set should result in a noticeable performance hit (compared to the previous two methods). No improvements to performance were considered here as accuracy was the primary concern.

4.5.4 Neural networks

The K-NN method is accurate but has the potential to be slow. Neural networks can be thought of as an approximation of *k*-nearest neighbour. They are usually faster, requiring only a couple of matrix operations, and contain more information than the simple statistical approach.

A multilayer perceptron based classifier was trained on 90% of the data and calibrated on the remaining 10%. Standard neural network evaluation techniques were applied to determine the best network configuration and learning parameters. This approach was taken on the advice of an expert in the field.

4.5.5 A hybrid approach

After evaluating the previous methods (see Section 4.6), a hybrid classifier was investigated. In a hybrid classifier, the results of two or more classifiers are combined to give a more reliable classification.

The ideal hybrid might be to identify the classifier that is most reliable for each part and take that probability. That is, to use the ad-hoc classifier for accompaniment and drums and the neural network for bass and lead. In theory, this should result in about 80% overall accuracy. Unfortunately, the outputs of the classifiers are not directly compatible. It may be that the probabilities of one classifier are all mainly above 50%, while another may calculate lower values. In the context of the other probabilities of a single classifier, a most likely classification can be determined. Other, less direct, methods must be employed.

It is also important to note that the relative values of some classifiers can not be relied upon. A good example of this is the neural network classifier. It was trained here to produce one outcome, that the classification with the highest probability is the correct one. It was not trained to rank the possible classifications. This means that all but the most likely part type must be ignored, the absolute values of none of the classifiers are important. It should be noted that this is not a limitation of neural networks in general, but does illustrate the problems associated with handling the probabilities directly.

The first method of integration considered was to weight each of the probabilities according to how reliable that value is. The probability for each part is determined by calculating the mean value across all of the classifiers. For example, the ‘bass’ probability from the neural network classifier is weighted by 0.8725. The same is done for the other classifiers and the mean value taken as the likelihood of the track being the bass line. As it uses the classifier outputs directly, even though they are weighted, it should be susceptible to the problems outlined above.

Using a voting arrangement overcomes the problems associated with handling probabilities directly. Each classifier is given one vote as to which part type it thinks is most likely. This technique can be modified so that the more reliable classifiers are given more votes than less reliable ones.

4.6 Comparison of accuracy

The manual classification was compared against that given by each classifier (see Table 6). The results were broken down to identify which method performs best for each part type. The mismatches were also broken down, this time to show where each method gets confused - when a classification is wrong, what is it mistaken for? Both Euclidean and 'city block' distance metrics were evaluated.

Note that a high accuracy for a given type can only be taken to show reliable classification if it is not often falsely assigned. A high false match would indicate behaviour similar to always assigning a track as bass, so giving a 100% record for determining bass tracks, but gives a poor overall performance rating.

The K-NN classifier was given 90% of the training set, so that 10% of the training set could be used to evaluate its performance, allowing a more direct comparison with the other methods.

Table 6 - Overall classifier accuracy

Classifier	Euclidean	City block
Statistical	58.88 %	66.17 %
Ad-hoc	71.21 %	
K-NN	67.27 %	72.73 %
Neural Network	75.23 %	
Hybrid	78.18 %	

The most accurate hybrid classifier used a 'one classifier, one vote' mechanism (78.18 %). Weighting the voting reduced the overall accuracy to 74.55 %. Weighting each of the probabilities and taking the average resulted in 76.36 % of tracks being successfully classified.

4.6.1 Statistical template classification

The statistical templates were tested against the set of tracks used to calculate the templates. The results are given in Table 8.

4.6.2 Ad-hoc classification

The ah-hoc classification was tested against the marked up set of tracks. The results are given in Table 7.

4.6.3 K-nearest neighbour

The *k*-nearest neighbour method was only tested with tracks not used in the feature vector table. This was to allow a more direct comparison with the simple template approach, although the test set was only a subset of that used for evaluating the other methods, so the numbers are not as reliable. The results are given in Table 9.

4.6.4 Neural network classification

The neural network classifier was tested against the marked up set of tracks. The results are given in Table 10.

4.6.5 Hybrid classification

The hybrid classifier was tested against the set of tracks used to test the *k*-nearest neighbour classifier. As KNN classification is part of the hybrid classifier, the observations made for testing the KNN classifier also apply here. The breakdown of results for the simple voting hybrid is given in Table 11.

Table 8 - Breakdown of statistical template classification

Part	Correct		Class when wrong	
	Eucl.	City	Eucl.	City
Acc	40.70 %	53.29 %	22.18 %	27.62 %
Bass	76.70 %	80.58 %	32.73 %	34.81 %
Drums	60.42 %	62.50 %	4.55 %	3.87 %
Lead	63.97 %	73.53 %	34.55 %	33.70 %
Overall	58.88 %	66.17 %		

Table 7 - Breakdown of ad-hoc classification

Part	Correct	Class when wrong
Acc	59.21 %	40.91 %
Bass	80.58 %	24.03 %
Drums	94.44 %	1.30 %
Lead	52.94 %	33.77 %
Overall	71.21 %	

Table 9 - Breakdown of *k*-nearest neighbour classification

Part	**Correct**		**Class when wrong**	
	Eucl.	**City**	**Eucl.**	**City**
Acc	40.00 %	53.33 %	33.33 %	46.67 %
Bass	84.62 %	84.62 %	11.11 %	6.67 %
Drums	90.00 %	90.00 %	5.56 %	6.67 %
Lead	64.71 %	70.73 %	50.00 %	40.00 %
Overall	67.27 %	72.73 %		

Table 10 - Breakdown of neural network classification

Part	**Correct**	**Class when wrong**
Acc	49.67 %	24.24 %
Bass	87.25 %	12.88 %
Drums	88.73 %	8.33 %
Lead	80.88 %	50.76 %
Overall	75.23 %	

Table 11 - Breakdown of hybrid classification

Part	Correct	Class when wrong
Acc	46.67 %	16.67 %
Bass	84.62 %	0.00 %
Drums	90.00%	0.00 %
Lead	94.12 %	83.33 %
Overall	78.18 %	

Table 12 - Comparison of classifier execution time

Classifier	Execution time	
	Euclidean	City block
Statistical	28.49 ms	28.48 ms
Ad-hoc	28.52 ms	
K-NN	32.69 ms	31.37 ms
Neural network	28.60 ms	
Hybrid	116.12 ms	

4.7 Comparison of speed

Although speed of classification was not the primary concern here, and the classifiers were not optimised for speed (other than the automatic optimisations performed by the compiler), it is interesting to note.

Timing analysis was performed by reading the set of MIDI files used to train the classifiers into memory. These files were then classified as many times as possible in ten minutes. The tests were performed on a single user 500 MHz Intel Pentium III running Microsoft

Windows 2000. The results, given in Table 12, show that the *k*-nearest neighbour classifier is not noticeably slower than the statistical approach. The hybrid classifier takes four times as long as any other classifier, which is not surprising given that it uses each of the others.

4.8 Evaluation of data filtering on a music database

One of the motivations for classifying the tracks in MIDI files was to improve the quality of data held in a music database. The effect of this was evaluated by building a contour database consisting of a collection of 870 MIDI files and the 100 files which were marked up when creating the training set. The database was of the type described in Chapter 3 and used an atomic length of 14. Secondary contours were used when both building and querying the database.

Only accompaniments and lead parts were added. Ideally, only lead parts would be added, but the classification systems easily confused lead with accompaniments, so both lead and accompaniment parts were used. The city block metric gave the best performance, so this was the one used for all classifiers. The resulting performance of the classifiers is given in Table 13.

Table 13 - Comparison of classifier accuracy when considering three part types

Classifier	Performance
Statistical	82.24 %
Ad-hoc	88.97 %
K-NN	90.91 %
Neural network	90.62 %
Hybrid	94.55 %

Database queries were simulated by extracting a segment of a lead part from the training set to represent realistic queries, as they were manually classified. The set of 500 queries was the same as that used to evaluate the accuracy of the unclassified database, see Section 3.7 for more details. The rank of the file from which the query was taken (the target file) was noted, along with the total number of matches found.

The test was run with varying allowances for errors, *d*, in the query. Note that these errors were in addition to those that the pitch contour representation inherently supports (i.e. more than changes in the exact pitch of a note). Errors were not simulated in the query, so this assumes exact pitch contour recall on the behalf of the user.

The mode rank of targets in the tests was 1, i.e. the target of a search is more likely to appear as the top choice than at any other rank. While this was encouraging, this did not tell the whole story as, in the case of no classification and where $d = 2$, this accounts for less than 55% of the queries. Figure 4.1 shows that, for $d = 0$, 90% of queries will return the target file in the top 2 files even if a classifier is not used.

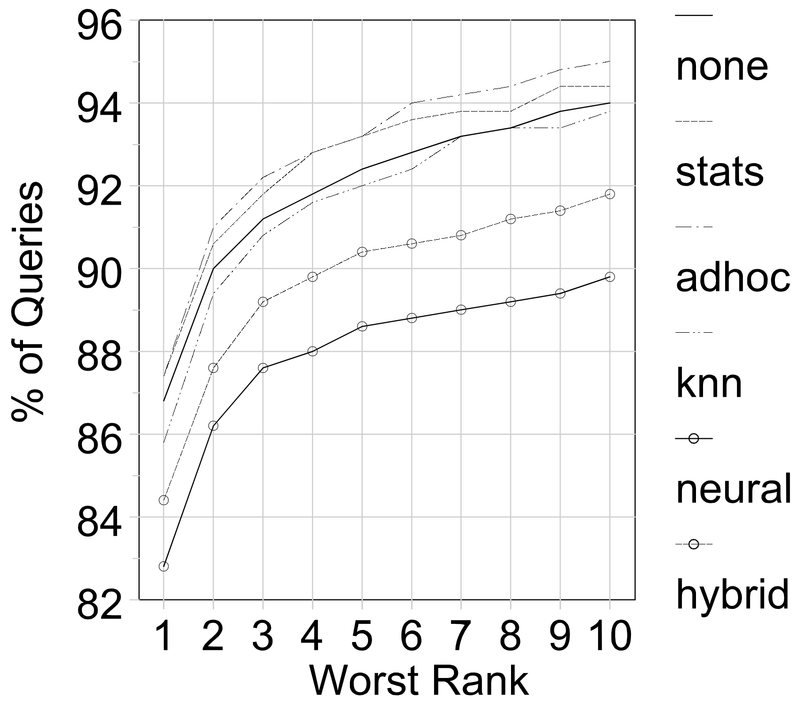


Figure 4.1 - Graph showing the ranking of queries for $d = 0$

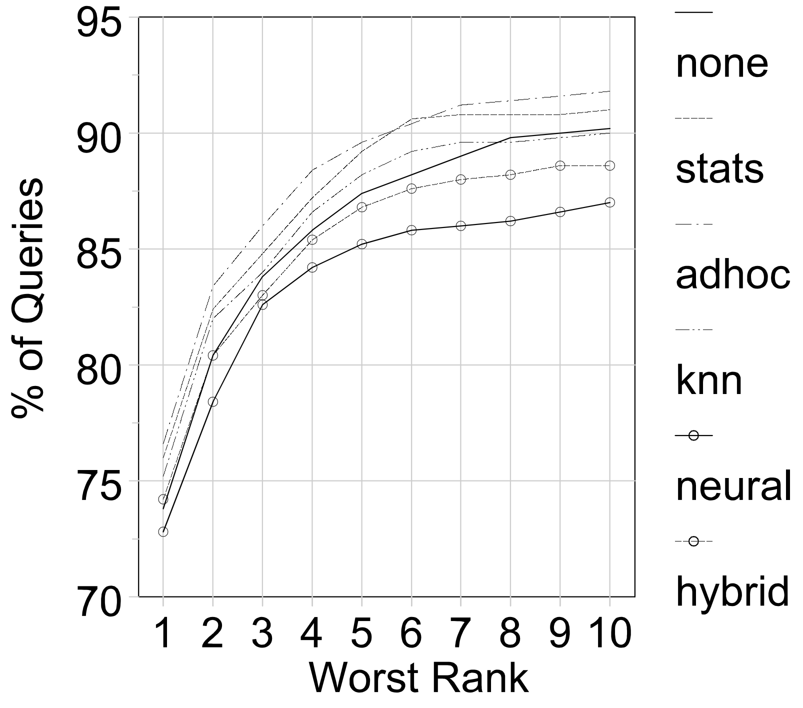


Figure 4.2 - Graph showing the ranking of queries for *d* = 1

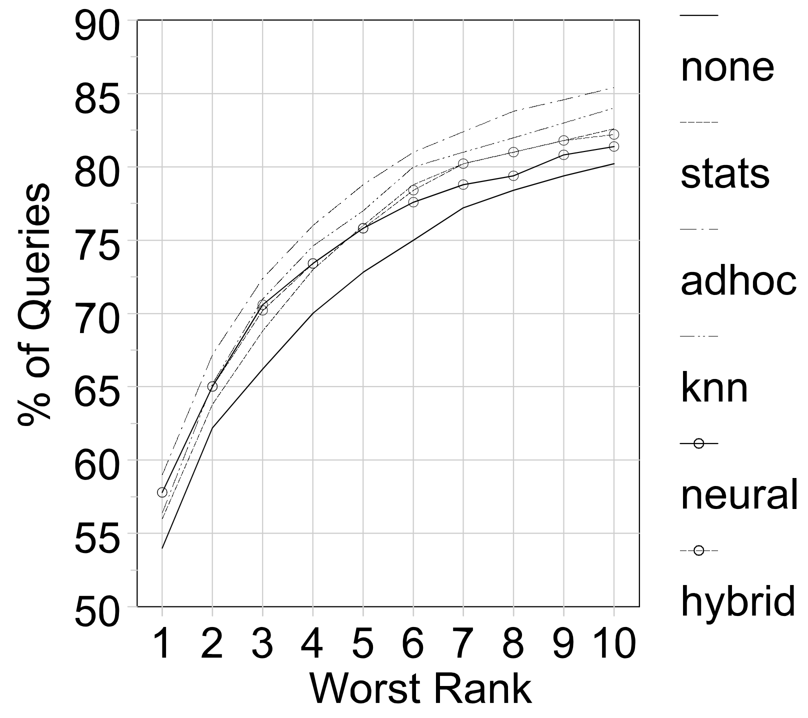


Figure 4.3 - Graph showing the ranking of queries for *d* = 2

Tables 14, 15 and 16 show the recall and the average rank and number of matches, taking into account any extreme results. It can be seen that the ad-hoc classifier is consistently the best, with nearly 100 % recall and giving the smallest number of matches. The difference between the classifiers becomes more apparent when the searches are approximate.

It is interesting to note that the classifier accuracy has little impact on the recall. Intuitively, better results should be obtained by using a more accurate classifier. The results of this evaluation suggest that the inverse is true, as recall is lower for the hybrid classifier (the most accurate) while the statistically classified database shows 100 % recall. The fact that the accuracy of a classifier is not reflected in the recall is probably due to themes being repeated by several parts in one song. That is, removing a part that contains a matching theme will reduce the weighting given to that song. In theory this should not matter, as the improved removal technique is applied to the whole collection, but it may be that this also results in one song looking much like another.

4.9 Conclusions

This chapter has shown the application of classification techniques to improve a music retrieval system. Some techniques are purely statistical while others attempt to bring to bear some knowledge of the nature of the content. It can be seen that classification of the musical part does improve matching, more notably when performing approximate searches.

The selection of an appropriate classifier will depend on the application. The hybrid classifier is the most accurate but it is also four times slower than the others. This chapter has also shown that, in the case of a retrieval system, the most accurate classifier is not always the most desirable. The ad-hoc classifier is a good all-round choice as it is relatively accurate, is one of the fastest to execute and was shown to be beneficial in a retrieval scenario.

The accuracy of the classifiers is shown to be extremely good when considering three types of musical part, as much as 94%. This is high enough for use in an automatic system, such as an automatic Internet indexing system or a content-based navigation system. Other types of classifiers may perform better on the difficult problem of differentiating between accompaniment and lead.

Table 14 - Ranking of targets using exact matching

Classifier	Found	Mean rank	Mean set size
None	100.0%	9.79	16.31
Statistical	100.0%	7.79	12.09
Ad-hoc	99.8%	6.91	11.47
K-NN	98.8%	7.21	11.73
Neural network	94.4%	6.98	11.54
Hybrid	96.6%	7.28	12.04

Table 15 - Ranking of targets allowing for one error in the query

Classifier	Found	Mean rank	Mean set size
None	100.0%	13.04	27.67
Statistical	100.0%	10.49	23.22
Ad-hoc	99.8%	9.16	20.51
K-NN	98.8%	9.78	20.96
Neural network	94.8%	9.30	20.52
Hybrid	97.0%	9.68	21.43

Table 16 - Ranking of targets allowing for two errors in the query

Classifier	Found	Mean rank	Mean set size
None	100.0%	23.32	91.35
Statistical	100.0%	18.99	76.88
Ad-hoc	99.8%	15.94	62.69
K-NN	98.8%	17.10	64.17
Neural network	94.8%	16.05	60.96
Hybrid	97.0%	16.92	63.88

5 Navigation of music

The previous chapters have demonstrated novel techniques for fast and accurate retrieval of musical content. Recall that the motivation for investigating and developing these techniques was to exploit the similarity between retrieval and navigation. This chapter considers the application of these techniques to navigation of music. The simpler case of temporal navigation, and the architecture developed to support it, is described first. The architecture is then extended to cater for content based navigation. The development and testing of a set of tools to provide proof of these concepts is also reported. This chapter presents an extension of the work reported in [Blackburn98].

5.1 Temporal navigation of music

It was explained in Section 2.2 that temporal navigation is the following of links that are based entirely on time information. Temporal navigation works with any temporal media, such as audio or video. The method of capturing of the time information used to resolve the link is application specific. The result of the capture will be either a time range or a point in time. The user interface will have a large impact on which representation is used, as a radio might use a single ‘link’ button while a computer-based archive viewer may allow the selection of a range. Whichever representation is used, the concept is the same. At any time whilst playing a piece of temporal media, a user may ask for a list of links related to that media.

A system which supports temporal navigation can be implemented with relative ease. It takes a time selection and something that identifies the media being played (perhaps a file name of the URL of a stream) and searches a database of links. This is compared with the link information in a link database to determine the list of links available to the user at that point. The information in the link database, which essentially consists of stream and position information in this case, is generated by the authors of links.

One scenario that uses temporal navigation is where a user is listening to a news broadcast. News broadcasts typically announce the headlines at the beginning of the programme, which is followed by the full stories. On hearing a headline of interest, a user may ask to jump to the full story for that headline without the need to listen to the other stories. Upon hearing a truncated interview, the user might then follow a link to the unedited interview.

The idea can also be applied to live streams, such as a radio broadcast. As live streams can be thought of as infinite, the timing information used for resolving links can not be based on a position in a stream. The nearest equivalent is to use the current time, i.e. the time at which the stream was broadcast. The operation of the link database is unlikely to be affected by this scenario, as it just has to resolve time periods into available links. The more troublesome implication is that the links will probably have to be created in realtime if it is a live performance. This is less of a problem when the 'live' broadcast is actually a pre-recorded program that is being transmitted on the stream.

Temporal navigation is similar to embedded hypertext (as found on the world wide web) in that links are not based on content but on the position in the document. This is useful for media files, such as digital audio files, as software to extract content from such files is not currently reliable enough for general use. Like embedded links, there is a considerable overhead in maintaining the linkbase.

5.1.1 System architecture

A system was developed which implemented temporal navigation of media documents on a Microsoft Windows 32-bit platform (Microsoft Windows NT v4). The architecture for the system is shown in Figure 5.1. A link-enabled application plays some media from a digital library. Upon the link functionality being invoked (perhaps through the use of a 'link' button) the application communicates the selected endpoint to the Link Manager. The Link Manager is a software component that persists for the duration of the user session. It uses one or more link services to resolve the endpoint against databases of links. The list of available links is then presented to the user.

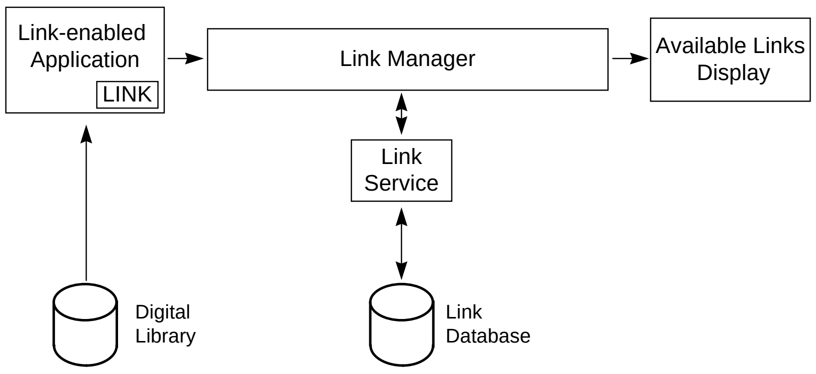
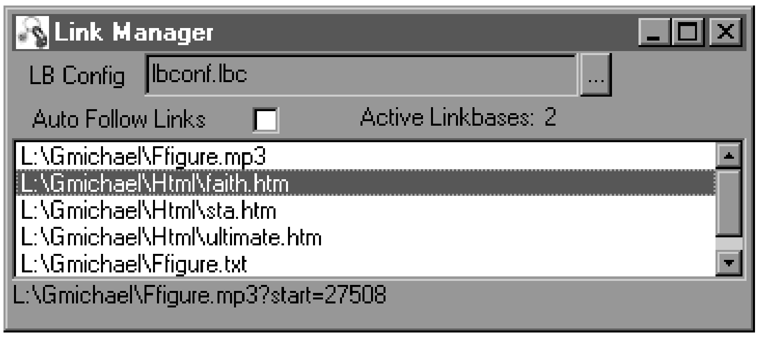


Figure 5.1 - The architecture used for temporal navigation

An endpoint identifies the document and a position or duration within the document. Endpoints are communicated between the components using a very simple message which is compliant with URL syntax. Links may consist of multiple source and destination endpoints; endpoints in links could carry alternative representations (e.g. text, thumbnails) so that it would be possible to navigate through the hyper-structure without accessing the temporal media at all. These formats were adopted for simplicity during development: in due course they could be adapted for interoperability with SMIL, XLink and XPointer.

5.1.2 The Link Manager



LB Config lbconf.lbc ...

Auto Follow Links ☐ Active Linkbases: 2

```
L:\Gmichael\Ffigure.mp3
L:\Gmichael\Html\faith.htm
L:\Gmichael\Html\sta.htm
L:\Gmichael\Html\ultimate.htm
L:\Gmichael\Ffigure.txt
L:\Gmichael\Ffigure.mp3?start=27508
```

Figure 5.2 - A screen shot of the Link Manger

The Link Manager component, shown in Figure 5.2, receives source endpoints from the other tools and resolves them using a link service. This link service could be a local linkbase or possibly a remote Open Hypermedia Protocol (OHP) compliant service. It also functions here as the available links display. Following a link is achieved by double clicking on the entry in the list or, if the 'Auto Follow Links' option is enabled, it will be followed as soon as it is resolved.

The software was developed to explore the concepts behind navigation; the rigorous design of a user interface was not considered and is beyond the scope of this thesis. For example, displaying the source endpoint (at the bottom of the Link Manager window) was only intended for demonstration purposes; it is not envisaged that the user would need to see this information.

5.1.3 The Linked Player

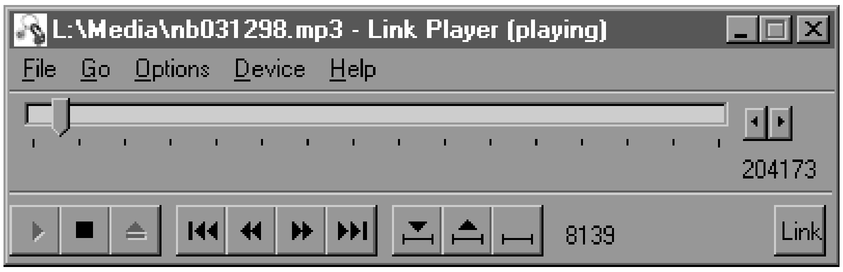


Figure 5.3 - A screen shot of the Linked Player playing a file

The Linked Player, shown in Figure 5.3, is a general purpose media player (essentially a wrapper for the operating system multimedia capabilities) that can send source endpoint information to the Link Manager, either automatically or as a result of user interaction. It is designed to resemble the original Media Player for Windows, a tool that many users are already comfortable with.

The player has several options which control when an endpoint is sent to the Link Manager for resolving. The ‘Link on Play’ option sends an endpoint whenever the play button is pressed. ‘Auto Link’ sends an endpoint more frequently, approximately every second. The size of the selection sent to the Link Manager may be changed by using a context menu on the link button.

5.1.4 Summary

The system was tested by following links from digital audio, video and MIDI files. The news broadcast scenario was implemented and run successfully. No link creation tools were developed due to the small scale of the experiment (i.e. just one person). This was not a problem for authoring one-off links, such as linking a single headline to a full story. Where the link management problem became noticeable was in creating links from multiple endpoints. A link was created from the chorus of a piece of pop music to a URL giving some information about that music. The link had to be authored for each occurrence of the chorus. This had to be repeated for two alternative arrangements of the same song.

The use of an external linkbase meant that the links could be administered without affecting the media that is the subject of the links. The advantage of this, over embedding the links in the document, was that all links could be listed instead of searching for them. This became important when preparing a demonstration of the software which required the locations of the media and the related links to move.

The system is useful in that it is a general approach that works for any time-based media. The disregard for content means that time is not spent extracting content which can be both difficult and time consuming. The problem of matching content based endpoints is avoided and is simply a matter of matching ranges of numbers (time stamps in this case).

5.2 Content based navigation of music

The previous section has shown that temporal navigation works but has its limitations. Each occurrence of the chorus in a piece of music had to be linked separately, even though they linked to the same information. It is desirable to be able to author a link on content, so that the content of the chorus is linked to some information, requiring only one link to be created and maintained. This section discusses the application of the principles of music retrieval to navigation. The development of a set of tools to navigate music is also described.

5.2.1 The system architecture

The temporal navigation architecture is extended to support content based navigation (CBN), as shown in Figure 5.4. Naturally, this requires content to be extracted from the media being played. A full representation of the content is neither required nor used, preferring to extract a set of features that support matching against features held in a linkbase.

One approach to CBN, and the one taken in the system developed, is to use the same link-enabled applications and to have the Link Manager call a separate component to extract the features. This has the advantage that the media player does not need to understand the content that it is playing. Indeed, the media player may not be aware of (or have access to) the underlying content, as is the case with the Linked Player. This also allows temporal links to be used along side content based ones, as the temporal information is passed to the Link Manager in the same way for both temporal and content based navigation.

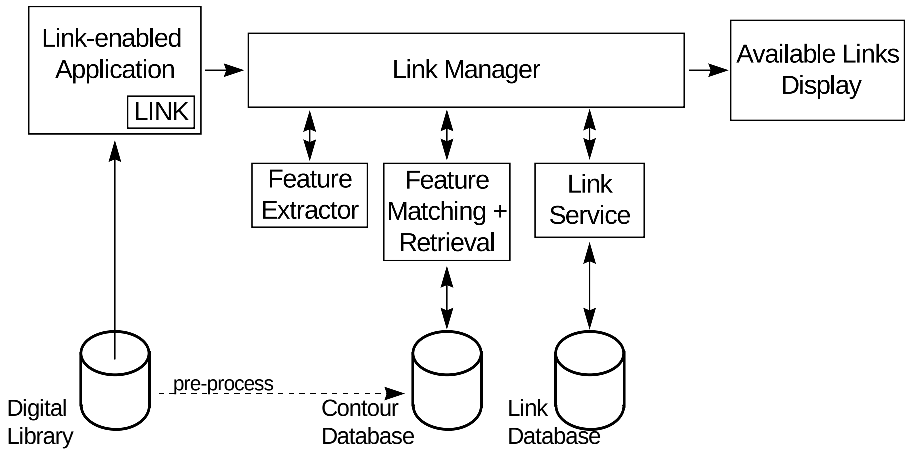


Figure 5.4 - The architecture extended for content based navigation

In some applications, there are alternatives to extracting the features at the Link Manager. If the media player deals directly with the content being played, the features could be extracted by the player. Extracting the features here, or converting the content into a more suitable representation than the raw media, saves time as the media does not need to be accessed twice (i.e. once to play it and once to extract features). This does have disadvantages, one of the more subtle is that passing a segment purely as content to the Link Manager removes all context associated with that segment. This may not matter for primitive content based links but precludes the use of intelligent link matching that could disambiguate features, a textual analogy might be the word ‘bear’. The meaning of the word ‘bear’ depends on the context associated with it. Removing this context forces the link to be applied blindly and does not allow the link author to specify the noun ‘bear’ as the source endpoint of a link.

Both this scenario and the one implemented presume client side feature management is desirable, as the Linked Player and Link Manger operate on the user’s machine. This also assumes that it is practical to extract features on the client machine, i.e. it is fast enough to perform the extraction. The feature extraction could instead be performed remotely, perhaps by dedicated extraction services, or in the link services themselves. This allows the feature extraction to be platform independent, which may help the developers of feature extraction components as developing for one platform will support all others. It does not require the client machine to be powerful, or for the media player to know about the content. Also, in the case of extraction at the link server, it does not require the Linked Player to understand how to convert a media segment into features appropriate for each server. As each link server may employ different representations for linkbases, even for similar media, this can be a big advantage. Remote extraction is one of the most general solutions and would also be the obvious choice except for one negative point. Each service that extracts features must obtain a copy of the media clip from which to process. This can be a large amount of data in the case of complex media, such as video, which requires a network with a high bandwidth. The media must also be accessible by the server, which excludes all media on a standard client machine that does not make all local files available via a web server (or some other suitable mechanism, e.g. FTP). This general solution also places a large strain on the servers, as features must be extracted for each link resolution. Scaling this approach to handle thousands of users is also troublesome.

There is no obvious right solution to the problem of where to extract the features. The approach chosen for the experiments was probably appropriate, given that the Linked Player has no knowledge of the content and that the link service was implemented as a process local to the client machine.

5.2.2 Adapting the contour database for navigation

The pitch contour retrieval database, described in the previous chapters, currently returns a file name when it is given some musical content. Recall that the difference between retrieval and navigation is that files *containing* some content are returned in the case of retrieval but that files (or URLs) *relating* to some content are returned in the case of navigation. The file is stored as a string in each case, so it is clear that the structure of the database does not need to be modified, only the procedure that associates the name of a file with its content.

The software tools were extended to allow an arbitrary string to be associated with some content. The path of the file containing the content was still retained, as it might be needed by some query validation process. An additional file path could also be stored. This was intended for associating musical content, such as a MIDI file, with the file from which the content was extracted. This is to support scenarios where navigation is performed from the soundtrack of a video.

Given the higher quality of data held by the database and, by inference, a smaller number of entries to index, the process may be enhanced for navigation. Firstly, the accuracy should be improved, so the atomic length can be reduced to decrease the size of the lookup table. Secondly, the ID referred to in the contour database may reference a structure more complex than just a filename. It could contain the full contour, which may be checked against the query contour.

The music linking example given in Section 5.1.4 was used as a proof of concept. The content of the chorus of Axel F. (by Harold Faltermyer) was linked to a URL of a web page discussing the film Beverly Hills Cop, from which the music is taken. Only one link was authored which was enough to associate all occurrences of the chorus with the URL, in all arrangements of the song.

This is the first known implementation of a content based navigation system for music[7]. The use of the indexing techniques from previous chapters ensures that it is fast enough for an interactive system. The time taken to resolve a query will be identical to the retrieval timings given in Section 3.8 plus the overhead of extracting the content.

A detailed evaluation of the effectiveness of the music database for navigation is not currently possible. A realistic evaluation requires a real linkbase to evaluate. The tools developed were not intended to be used as part of a user trial as it is out of the scope of this thesis; most of the links must be manually entered into the linkbase. Due to the lack of a suitable linkbase, no evaluation was attempted.

[7]While MAVIS supported content based navigation of digital audio the content representation (FFTs) made it impossible to replicate the scenario given here. For one link to work across several arrangements requires the endpoint of the link to sound nearly identical (same combination of instruments played the same way).

5.3 Notes on implementation

The approach to navigation taken in this thesis, namely to extend retrieval techniques, may appear to be so natural a route as to be inconsequential. This is misleading as developing any system for navigation of audio, content based or otherwise, is not a trivial task. It is only because of the approach taken that navigation would appear to be the logical next step from retrieval. This section gives some insight into the scale and structure of the software developed.

The retrieval engine described in the previous chapters was written in C++ using the Borland C++ Builder development environment. The core functionality was developed as a set of libraries that could manipulate MIDI files, create and use an index file and finally manage the not-so-small task of query processing and result merging. The set of libraries consist of 25,000 lines of source code in total (including comments and white space). The Microsoft Windows specific code adds several thousand lines to this count (for just the retrieval tools).

The majority of the audio navigation system, detailed in this chapter, is designed purely for the MS Windows platform. This is a further 12,000 lines of source code, although that includes basic XML parser and support for the Open Hypermedia Protocol (OHP).

The navigation system implementation closely follows the architecture diagrams given earlier. The Link Player uses a COM (Common Object Model) object to communicate endpoints to the Link Manager. (Incidentally, the use of COM has the side effect of being usable from many scripting languages.) A link server is implemented as a COM object, which allows new servers to be implemented without modifying the code. The Link Manager can be configured to use different link servers and linkbases. The feature extraction is currently embedded in the Link Manager, although this could also be modularised in a similar manner.

5.4 Using classification in navigation

Classification has uses in both temporal and content based navigation, although one could argue that the temporal case is a form of the latter as the temporal location of content can be determined. The classification could be used as a parameter to link resolution. This enables temporal links to be authored on particular parts of the music, i.e. from 20 seconds to 30 seconds of the bass line. Similarly, content based navigation would allow an anchor to take the form of ‘lead lines that sound like this’.

Reliable classification can be used to enhance a content based navigation system by allowing more precise (but still general) links to be authored. For example: ‘bass lines like this one are linked to some information’. The recall accuracy of a link database when used in conjunction with part filtering has yet to be evaluated. It should prove to be at least as useful as for retrieval, due to the fact that only a subset of the music collection is stored. Implementing this concept would require the ID given in the database to refer to a structure contain the additional conditions of the link applying, such as valid part types. This was not investigated further as it is an extension of content based navigation and so is not directly the focus of this thesis.

5.5 Conclusions

This chapter has shown how musical pitch contours can be used as features for content based navigation. An extensible architecture for temporal navigation has been shown to be effective. The architecture has been extended to work with content of a restricted set of media types (i.e. MIDI files) to provide content based navigation. Advantage was taken of the similarity between content based retrieval and navigation by utilising the same content matching techniques in both cases. The content based navigation system described is the only one known to support music at the score level.

Early versions of these tools formed part of a technical demonstration given at ACM Multimedia ‘97 in Seattle. Since then, further requirements of a useful CBN system have discovered as a side effect of developing the prototype tools. For example, a history list was added to the media browser, as is now common among web browsers and so expected of any competent navigational system.

6 Conclusions

This thesis has investigated the content based retrieval and navigation of music. It has presented novel techniques for fast retrieval and architectures, and prototype tools, for navigation. These elements have been drawn together to present the first known system capable of navigating music by its content. The melodic pitch contour has been shown to be a powerful and flexible representation of this content.

It has presented a fast method of retrieval if music which uses contour *n*-grams to index a collection of MIDI files. Approximate lookup of the index is supported through the use of query expansion. Errors can be simulated in short queries, while long queries may be split to avoid errors. The use of a custom database format has been shown to be the fastest, although the text retrieval engine is not too slow.

A new representation, secondary contours, was introduced as a complementary representation that does not require the development of additional indexing techniques. This was shown to improve accuracy while adding little overhead to the search process. Using a secondary contour in parallel with a primary one was shown to be more powerful than using a representation with five pitch classes.

The results of the index lookup must undergo several stages of verification before a concrete list of matches can be presented to the user. In practice, the final check against the source file is not usually required. The use of secondary contours as an additional representation proves successful.

The handling of long queries is currently optimised for fast index lookup but this can result in a trade off against the accuracy of an index only lookup and can actually take longer, as there are more candidate files.

Chapter 4 explained the verbose nature of pitch contours in MIDI files and showed that filtering of content is required for large databases. Methods for the classification of musical parts (or roles) were developed and evaluated in the context of retrieval. Some techniques were purely statistical while others attempted to bring to bear some knowledge of the nature of the content. It could be seen that classification of the musical part does improve matching, more notably when performing approximate queries. The ad-hoc method is the most accurate and is also the fastest.

Musical pitch contours were shown to be useful features for content based navigation in a prototype system, which is believed to be the first of its kind. An extensible architecture for temporal navigation was shown to be effective. The architecture was extended to work with the content of a restricted set of media types (i.e. MIDI files) to provide content based navigation. The similarity between content based retrieval and navigation was utilised and the same content matching techniques were used in both cases.

6.1 Future work

While this thesis has focussed on its particular contribution, the work has raised some interesting questions and directions for research stemming from this.

6.1.1 Developing and evaluating a real linkbase

The nature of the data indexed in a linkbase is likely to be different from that in an index intended for retrieval, which may have consequences for the indexing technique used. This can not be proved until real linkbases are developed. This thesis has proved the feasability of temporal and content based navigation of music. The next step is to develop a set of tools that allow non-technical users to create and manage linkbases. Only then can a detailed evaluation be performed.

6.1.2 Notational differences for matching

Due to MIDI being a performance oriented standard, key presses rather than notes are communicated. This means that $C^{\#}$ and D^{b} are transmitted as the same key on a piano (which they are). The difference is not important to the listener but it is to musicologists. Although similarity matching in music concentrates on pitch, comparison of the notation might allow better use of the existing data. As MIDI files are the most widely available sources of music, some method of determining the correct pitch notation will be required. This issue has already been investigated by Blombach [Blombach95] who has developed an algorithm for achieving this. This would give five extra pitch classes per octave, taking the total number from twelve to seventeen.

This thesis has not evaluated this approach, as melodic pitch contours would ignore this additional information. Similarly, pitch contours could, in principle, be applied to microtonal music, or indeed and scales other than the standard 12-note equally tempered scale.

6.1.3 Improving retrieval

It is clear that the accuracy of the retrieval technique will play a large part in the usability of a system for navigation. The indexing methods presented in this thesis are believed to be suitable for use in a linkbase. Currently, 85% of queries result in the target song appearing in the first four matches when there are 7,600 complete songs in the database. The level performance will drop as more files are added, eventually dropping to an unacceptable level. There are a number of improvements that could be made to the retrieval system described in this thesis.

Scale the contour counts. The current implementation notes the number of occurrences of each sub-contour in a file on adding it to the database. This is an absolute number which anecdotal evidence suggests can cause problems. This is because longer or more complicated pieces of music will naturally contain more sub-contours. This causes the system to rank larger MIDI files higher than smaller ones. This is especially evident when locating different arrangements of the same song. Normalising the sub-contour count on a per file basis, so that the count reflects the relative relevance of that contour for that file, should stop this. Normalisation has the consequence that one stage of the results checking, ensuring enough sub-contours per match, can no longer be performed.

Add tracks individually. Stephen Downie showed that his database did not need to store the precise location of contours, only the name of the file that they occur in. The accuracy was only marginally improved by ensuring matches were complete. The same is probably not true for adding an entire MIDI file as one entity, as there are many more sub-contours per file than per track. Adding the tracks in a MIDI file to the database one at a time is likely to improve accuracy overall. This concept was not examined further as links were implemented with a single contour as the source of the link.

Using rhythm contours. The use of rhythm contours has been reported by many people in the field to be beneficial to retrieval. The improvement has not been as great as adding more pitch classes but is still worth considering. One of the advantages with adding support for rhythm contours to the system described in this thesis, is that it can make direct use of the underlying contour database. This would be similar to the way in which secondary contours are currently supported.

Quantize the MIDI before extracting pitch contours. Recall that MIDI is a performance oriented standard as a musician is unlikely to play a piece of music exactly like it is written down on paper. Because of this, and the limitations of MIDI, the timing of notes in a MIDI file will be slightly out (when compared with the musical score). This can result in three notes, that appear to a listener to sound simultaneously, starting at slightly different times. This will have an impact on the contour generated from that performance, as what should be one pitch transition is recorded as several. Most MIDI sequencers allow a musician to tidy the note timings (quantize), which would stop this from happening, but some feel that this can remove the feeling from a piece and make it sound computer-generated. As it is likely that many files will not be quantized, it may be beneficial to a publicly used system.

Extend the system to support n-ary contours. This thesis has shown the use of secondary contours to be a highly beneficial in retrieval, improving average results by as much as 40%. It has also shown the expressiveness of the contour combination to be comparable with a five-pitch representation. Other researchers in the field have proved that more accurate results are obtained by using a representation with more pitch classes. Considering all these points, it is reasonable to expect that using a tertiary contour should improve the current system. This could be further extended to *n*-ary contours, although there may be a diminishing return, in terms of space vs. improvement of accuracy.

Improve the presentation of match relevance to the user. The scores are currently absolute, even if the contour count is normalised per track or file. This can confuse users as queries of similar lengths can return wildy different scores, largely due to the type of query expansion used. This issue can be resolved by either presenting the scores in a normalised form (i.e. the top-ranking match is given a mark of 1000 and the others given marks relative to that) or by not displaying the scores to the user and just showing the ranking position of each match. The former is not ideal, as it suggests a 100% match for the top-ranking item (which is not always the case). The later is no better, as all relative match information is lost, although it is arguably not misleading (due to no claim of a 100% match). Obviously, the ideal score display should show both the accuracy of each match and how close the matches are. No solution was investigated because this not an issue with the implemented linkbase, as all matching links are displayed.

6.1.4 Texture of music

It would be interesting to see if determining the ‘texture’ of music, through the use of co-occurrence matrices (Haralick [Haralick73]) normally used to match texture in images, could be used to retrieve links in a linkbase, or as an aid to classification. Co-occurrence matrices for music would give the probability of the next note being *x*, given that the current note is *y*. So if a tune consisted of two alternating notes then the musical equivalent of stripes would be represented. It is unlikely that texture alone would be enough to identify content, as it is a very abstract concept, but could be useful in reducing the search space.

Applying the technique in reverse would also be intriguing. That is, given a texture matrix, compose a piece of music using these probabilities. However, the value of any results is of questionable value to the field of computer science.

6.2 Going beyond content

This thesis has shown that the concept of using musical content as an endpoint in a navigation system is a viable one. The application of techniques to further understand the content, with a view to improving both retrieval and navigation, has also been discussed (i.e. musical parts). As the size and use of musical retrieval and navigation systems grows, it will become increasingly important to understand and exploit the semantics of music.

A content based link currently matches as long as the notes are similar enough (for musical links). The ability to comprehend what the composer intended to convey with a given set of notes gives much more precise control. It would make it possible to author links from musical constructs (e.g. a chorus) or even intended mood (e.g. sorrowful, happy). This might be described more generally as the context of the link. The development of a musical semantic engine poses the biggest challenge in this field, although it also promises the biggest rewards.

Bibliography

[Beeferman97] Beeferman, D., QPD: Query by Pitch Dynamics - Indexing Tonal Music by Content, Project Report, Link Group, Carnegie Mellon University, December 4th 1997.

[Blackburn98] Blackburn, S. and De Roure, D., A Tool for Content Based Navigation of Music, in Proc. ACM Multimedia '98, 1998, pp. 361-368.

[Blackburn2000] Blackburn, S. and De Roure, D., Musical Part Classification in Content Based Systems, Proceedings of the OHS 6 and SC 2. Published in Lecture Notes in Computer Science, (LNCS 1903), Springer-Verlag, 2000, pp. 66-76.

[Blombach95] Blombach, A.K., Determining Keys and Correct Pitch Notation in Tonal Melodies, Computers in Music Research, Volume V, Spring 1995, pp. 67-102.

[Buchanan92] Buchanan, M.C. and Zellweger, P.T., Specifying Temporal Behaviour in Hypermedia Documents, in Proc. ECHT ‘92 European Conference on Hypertext, November 1992, pp. 262-271.

[Bulterman98] Bulterman, D.C.A., Hardman, L., Jansen J., Mullender, K.S. and Rutledge, L., GRiNS: A GRaphical INterface for Creating and Playing SMIL Documents, in Proc. 7th International World Wide Web Conference, Brisbane, Australia, April 1998, pp. 519-529.

[Bush45] Bush, V., As We May Think, Altantic Monthly, July, 1945, pp 101-108.

[Carr95] Carr, L., De Roure, D., Hall, W. and Hill, G., The Distributed Link Service: A Tool for Publishers, Authors and Readers, in Proc. Fourth International World Wide Web Conference: The Web Revolution, Boston, Massachusetts, USA, December 1995, pp. 647-656.

[Chen] Chen, A.L.P., Liu, C.C., Hsu, J.L., Efficient Near Neighbor Searching Using Multi-Indexes for Content-Based Multimedia Data Retrieval, *to appear in* Multimedia Tools and Applications, Kluwer Academic Publishers.

[Chen98a] Chen, A.L.P., Chen, J.C.C., Query by Rhythm An Approach for Song Retrieval in Music Databases, in Proc. of IEEE International Workshop on Research Issues in Data Engineering, 1998, pp. 139-146.

[Chen98b] Chen, A.L.P., Hsu, J.L., Liu, C.C., Efficient Repeating Pattern Finding in Music Databases, in Proc. ACM 7th International Conference on Information and Knowledge Management, November 3 - 7, 1998, pp. 281-288.

[Chen99] Chen, A.L.P., Liu, C.C., Hsu, J.L., Efficient Theme and Non-Trivial Repeating Pattern Discovering in Music Databases, in Proc. 15th IEEE International Conference on Data Engineering, 1999, pp. 14-21.

[Davis92] Davis, H., Hall, W., Heath, I., Hill, G., and Wilkins, R., Towards an Integrated Information Environment with Open Hypermedia Systems, in Proc. Fourth ACM Conference on Hypertext, Models for Open Systems, 1992, pp. 181-190.

[Davis96] Davis, H., Lewis, A. and Rizk, A., OHP: A Draft Proposal for a Standard Open Hypermedia Protocol, in Proc. 2nd Workshop on Open Hypermedia Systems. UCI-ICS Technical Report 96-10, University of California, Irvine, pp. 27-53.

[DeRoure] De Roure, D., Blackburn, S., Content based navigation of music using melodic pitch contours, Multimedia Systems, Vol. 8, No. 3, ACM Springer-Verlag, October 2000, pp. 190-200.

[DeRoure96] De Roure, D., Carr, L., Hall, W., and Hill, G., A Distributed Hypermedia Link Service, in Third International Workshop on Services in Distributed and Networked Environments, Macau, June 1996, pp. 156-161. IEEE.

[DeRoure98a] De Roure, D. and Blackburn, S., Amphion: Open Hypermedia Applied to Temporal Media, in Proc. 4th Open Hypermedia Workshop, pp. 27-32, 1998.

[DeRoure98b] De Roure, D., Blackburn, S., Oades, L., Read, J., Ridgway, N., Applying Open Hypermedia to Audio, in Proc. Hypertext 98. 1998, pp. 285-286.

[DeRoure99] De Roure, D., El-Beltagy, S., Blackburn, S. and Hall, W., A Multiagent System for Content Based Navigation of Music, in Proc. ACM Multimedia 99 (Part 2), 1999, pp. 63-66.

[Deutsch98] Deutsch, W.A., Visualisation of Music Signals, presented at ACM Multimedia workshop on Content Processing for Music, September 1998. Available at http://www.acm.org/sigmm/MM98/aigrain.html [September 2000].

[Dowling78] Dowling, W. J., Scale and Contour: Two Components of a Theory of Memory for Melodies, Psychological Review 85, 1978, pp. 341-354.

[Downie99] Downie, S., Evaluating a Simple Approach to Music Information Retrieval: Conceiving Melodic N-Grams as Text, PhD Thesis, Faculty of Information and Media Studies, University of Western Ontario, Ontario, July, 1999.

[Dunn99] Dunn, J.W., Mayer, C.A., VARIATIONS: A Digital Music Library System at Indiana University, in Proc. ACM Digital Libraries ‘99, 1999, pp. 12-19.

[Foote97] Foote, J.T., Content-based Retrieval of Music and Audio, in Proc. SPIE ‘97, 1997, pp. 138-147.

[Foote99] Foote, J.T., An Overview of Audio Information Retrieval, Multimedia Systems 7, 1999, pp. 2-10.

[Fountain90] Fountain, A.M., Hall, W., Heath, I., and Davis, H.C., MICROCOSM: An Open Model for Hypermedia with Dynamic Linking, in Proc. ECHT ‘90 European Conference on Hypertext, Building Hypertext Applications, 1990, pp. 298-311.

[Fushikida98] Fushikida, K. Hiwatari and Y. Waki, H., A Content-Based Video Retrieval Method Using a Visualized Sound Pattern, in Proc. Visual Database Systems, 1998, pp. 208-213.

[Ghias95] Ghias, A., Logan, J., Chamberlin, D. and Smith, B. C., Query by Humming - Musical Information Retrieval in an Audio Database, in Proc. Multimedia ‘95, San Francisco, California, November 1995, pp. 231-236.

[Gibson99] Gibson, D., Name That Clip: Content-based Music Retrieval, in Proc. The Exploratory Workshop on Music Information Retrieval Workshop, SIGIR99, Berkeley, California, August 1999, pp. 6.

[Goose95] Goose, S. and Hall, W., The Development of a Sound Viewer for an Open Hypermedia System, in The New Review of Hypermedia and Multimedia, Vol. 1, 1995, pp. 213-231.

[Hall94] Hall, W., Ending the tyranny of the button. IEEE Multimedia, Vol. 1, No. 1, Spring 1994, pp. 60-68.

[Haralick73] Haralick, R.M., Shanmugam, K., and Dinstein, I., Textural Features for Image Classification. IEEE Transactions on Systems, Man and Cybernetics 3, 1973, pp. 610-621.

[Hardman94] Hardman, L., Bulterman, D.C.A. and Rossum, G.v., The Amsterdam Hypertext Model: Adding Time and Context to the Dexter Model, Communications of the ACM, Vol. 37, No. 2, February 1994, pp. 50-62.

[Hewlett92] Hewlett, W.B., A Base-40 Number-Line Representation of Musical Pitch Notation, Musikometrica 50 / 4, 1992, pp. 1-14.

[Hirata93] Hirata, K., Hara, Y., Shibata, N., Hirabayashi, F., Media-based Navigation for Hypermedia Systems, in Proc. ACM Hypertext ‘93, November 1993, pp. 159-173.

[HTML4] HTML 4.01 Specification, World Wide Web Consortium (W3C), 24 December 1999. Available at http://www.w3.org/TR/html4 [September 2000].

[Huron97] Huron, D., “Humdrum and Kern: Selective Feature Encoding” in Beyond MIDI: The Handbook of Musical Codes, Cambridge: MIT Press, 1997, pp. 375-401.

[HyTime] ISO/IEC 10744, Hypermedia / Time-based Structuring Language (HyTime), International Organization for Standardization (ISO), Geneva, Switzerland, 1997.

[Kimber96] Kimber, W.E, Practical Hypermedia: An Introduction to HyTime, Prentice Hall, 1996.

[Kornstadt98] Kornstadt, A., Themefinder: A Web-based Melodic Search Tool, Computing in Musicology, Vol. 11, 1998, pp. 231-236.

[Lap] Lap, Y.C., Catfind, University of Hong Kong. Available at http://zodiac.csis.hku.hk:8192/catfind/Music/ContentSearch.html [30 July 2000]

[Lemstrom98] Lemström, K. and Laine, P., Musical Information Retrieval Using Musical Parameters, in Proc. 1998 International Computer Music Conference (ICMC '98), 1-6 October, 1998, pp. 341-348.

[Lemstrom2000] Lemström, K. and Tarhio, J., Searching Monophonic Patterns within Polyphonic Sources, in Proc. RIAO'2000 Recherche d'Informations Assistée par Ordinateur (Content-Based Multimedia Information Access). Paris, France, April 12-14, 2000, pp. 1261-1279 (Vol. 2).

[Levenshtien65] Levenshtien, V., Binary Codes Capable of Correcting Spurious Insertions and Deletions of Ones, Problems of Information Transmission, Vol. 1, 1965, pp. 8-17.

[Lewis96a] Lewis, P.H., Davis, H.C., Griffiths, S.R., Hall, W., and Wilkins, R.J., Media-based Navigation with Generic Links, in Proc. 7th ACM Conference on Hypertext, New York, 16-20 March 1996, pp. 215-223. ACM Press.

[Lewis96b] Lewis, P.H., Davis H., Dobie, M., Hall, W., Kuan, J., and Perry, S., Content Based Navigation in Multimedia Information Systems, in Proc. ACM Multimedia 96, New York, NY, USA, November 1996, pp. 415-416. ACM Press.

[Lindsay94] Lindsay, A., Using Contour as a Mid-level Representation of Melody, Masters Thesis, Massachusetts Institute of Technology, 1994.

[Liu97] Liu, C.C., Chen, A.L.P., Vega: A Multimedia Database System Supporting Content-Based Retrieval, Journal of Information Science and Engineering, Vol. 13, No. 3, September 1997, pp. 369-398.

[Martin96] Martin, K.D., Automatic Transcription of Simple Polyphonic Music: Robust Front End Processing, Technical Report 399, Massachusetts Institute of Technology, The Media Laboratory, December 1996.

[McNab95] McNab, R.J., Smith, L.A. and Witten, I., Signal Processing for Melody Transcription, Working Paper 95/22, Dept. of Computer Science, University of Waikato, New Zealand, August 1995.

[McNab96] McNab, R.J., Smith, L. A., Witten, I.H., Henderson, C.L. and Cunningham, S.J., Towards the Digital Music Library: Tune Retrieval from Acoustic Input, in Proc. Digital Libraries ‘96 (DL'96), 1996, pp. 11-18.

[McNab97] McNab , R.J., Smith L.A., Bainbridge, D. and Witten I.H., The New Zealand Digital Library MELody inDEX, Technical Report, D-Lib Magazine, May 15, 1997.

[McNab2000] McNab, R.J., Smith, L. A., Witten, I.H., Henderson, C.L., Tune Retrieval in the Multimedia Library, Multimedia Tools and Applications, Vol. 10, 2000, pp. 113-132.

[Melucci99] Melucci, M. and Orio, N., Musical Information Retrieval using Melodic Surface, in Proc. Digital Libraries ‘99, 1999, pp. 152-160.

[MIDI] MIDI Manufacturers Association (MMA), The Complete MIDI 1.0 Detailed Specification, v.96.1, MIDI Manufacturers Association, La Habra, CA, USA, 1996.

[Millard2000] Millard, D.E., Moreau, L., Davis, H.C. andReich, S., FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability between Hypertext Domains, in Proc. ACM Hypertext '00, May 30 - June 3, San Antonio, TX, 2000, pp. 93-102.

[MPEG-1] ISO/IEC 11172 (Parts 1-5), Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s (MPEG-1), International Organization for Standardization (ISO), Geneva, Switzerland, 1993-1999.

[MPEG-2] ISO/IEC 13818 (Parts 1-9), Generic Coding of Moving Pictures and Associated Audio Information (MPEG-2), International Organization for Standardization (ISO), Geneva, Switzerland, 1996-2000.

[MPEG-4] [ISO/IEC 14496 (Parts 1-6), Coding of Audio-visual Objects (MPEG-4), International Organization for Standardization (ISO), Geneva, Switzerland, 1999.

[MPEG-7] ISO/MPEG N3445, Overview of the MPEG-7 Standard, International Organization for Standardization (ISO), Geneva, Switzerland, May/June 2000.

[Nack99] Nack, F. and Lindsay, A.T., Everything You Wanted To Know About MPEG-7: Part 1, IEEE Multimedia, July-September 1999, pp. 65-77.

[Navarro98] Navarro, G., Approximate Text Searching, Ph.D. thesis, Dept. of Computer Science, University of Chile, December 1998.

[Nelson81] Nelson, T., Literary Machines, Published by the author, 1981.

[Ossenbrugen94] Ossenbruggen, J.v. and Eliëns A., Music in Time-based Hypermedia, in Proc. ECHT'94 European Conference on Hypermedia Technologies, Technical Briefings, 1994, pp. 224-227.

[Parsons75] Parsons, D., The Directory of Tunes and Musical Themes, Spencer Brown, 1975.

[Pikrakis96] Pikrakis, A., Theodoridis, S. and Kamarotos, D., Recognition Of Isolated Musical Patterns In The Context Of Greek Traditional Music, in Proc. IEEE ICECS ‘96, 1996.

[Prechelt99] Prechelt, L. and Typke, R., An Interface for Melody Input, Submitted to ACM Transactions on Computer-Human Interaction. December 1999.

[Ristad96] Ristad, E.S. and Yianilos, P.N., Learning String Edit Distance, Technical Report CS-TR-532-96, Department of Computer Science, Princeton University, 1996.

[Rolland98] Rolland, P.Y., Raskinis, G., Ganascia, J.G., 1999. Musical Content-based Retrieval: An Overview of the Melodiscov Approach and System, in Proc. ACM Multimedia 98, 1998, pp. 81-84.

[Salosaari98] Salosaari, P. & Järvelin, K., MUSIR - A Retrieval Model for Music, manuscript 13 p, Department of Information Studies, University of Tampere, Finland, 1998.

[Sawhney96] Sawhney, N. and Murphy, A., ESPACE 2: An Experimental HyperAudio Environment, in Proc. CHI '96, 1996, pp. 105-106.

[Scheirer99] Scheirer, E.D., Structured Audio and Effects Processing in the MPEG-4 Multimedia Standard, Multimedia Systems 7, 1999, pp. 11-22.

[SMDL] ISO/IEC 10743, Standard Music Description Language (SMDL), International Organization for Standardization (ISO), Geneva, Switzerland, 1998.

[SMIL] Synchronized Multimedia Working Group, Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, Technical Report REC-smil-19980615, World Wide Web Consortium (W3C), June 1998. Available as http://www.w3.org/TR/REC-smil/ [30 July 2000].

[SMIL20] Synchronized Multimedia Working Group, Synchronized Multimedia Integration Language 2.0 (SMIL 2.0) Specification, Working Draft, World Wide Web Consortium (W3C), 21 September 2000. Available at http://www.w3.org/TR/smil20 [September 2000].

[Sonoda98] Sonoda, T., Goto, M., Muraokal, Y., A WWW-based Melody Retrieval System, in Proc. International Computer Music Conference (ICMC), 1998, pp.349-352.

[Sonoda2000] Sonoda, T., Muraokal, Y., A WWW-based Melody Retrieval System, *to appear in* Proc. International Computer Music Conference (ICMC2000), 2000.

[Tseng99] Tseng, Y.H., Content-Based Retrieval for Music Collections, in Proc. Conference on Information Retrieval ‘99, 1999, pp. 176-182.

[Uitdenbogerd98] Uitdenbogerd, A.L. and Zobel J., Manipulation of Music for Melody Matching, in Proc. ACM Multimedia '98, 1998, pp. 235-240.

[Vanzyl94] Vanzyl, A., HyperScape: The Hypertext and Information Management Environment for the Macintosh, in Proc. ECHT'94 European Conference on Hypermedia Technologies, Demonstrations, 1994.

[Wiggins93] Wiggins, G., Miranda, E., Smail, A., Harris, M., A Framework for the Evaluation of Music Representation Systems, Computer Music Journal, Vol. 17, No. 3, Fall 1993, pp. 31-42.

[Wold96] Wold, E., Blum, T., Keislar, D. and Wheaton, J., Content-Based Classification, Search, and Retrieval of Audio, IEEE Multimedia, Fall 1996, pp. 27-36.

[Wu92] Wu, S. and Manber, U., Agrep - A Fast Approximate Pattern-matching Tool, in Proc. USENIX Technical Conference, 1992, pp. 153-162.

[Yip99] Yip, C.L. and Kao, B., A Study on Musical Features for Melody Databases, in Proc. 10th International Conference on Database and Expert Systems Applications, 1999, pp. 724-733.