A Tool for Content Based Navigation of Music

Steven Blackburn and David DeRoure
Multimedia Research Group, Department of Electronics and Computer Science
University of Southampton, Southampton SO17 1BJ, UK
+44 (0)1703 592418

{sgb97r, dder}@ecs.soton.ac.uk

1. ABSTRACT

This paper presents a system which employs the accepted notion of melodic pitch contours to support content-based navigation around a body of multimedia documents including MIDI and digital audio files. The system adopts an open hypermedia model which enables the user to find available links from an arbitrary fragment of a piece of music, based on the content or location of that fragment. The design of the tools, indexed contour database and the fast contour-matching algorithms are discussed.

1.1 Keywords

Open hypermedia, content based navigation, branching audio, melodic contours, pitch contours, query by humming

2. INTRODUCTION

Ghias *et al* [6] and McNab *et al* [11] describe systems for querying an audio database by acoustic input, such as humming the tune of a song. The query is pitch-tracked and represented as a sequence of relative pitch changes (a melodic *pitch contour)*, thus factoring out inaccuracies of timing and tuning; the contour is then matched against a database of songs, using an appropriate string matching technique. Hence these systems implement *content based retrieval* (CBR).

We extend this work with an alternative design for the contour database, which we have exercised with a database of around 8000 songs, and we then employ pitch contours in support of *content based navigation* (CBN), whereby the user can select arbitrary fragments in a piece of music and query for the available hypermedia links. In order to investigate content based navigation we have created a suite of tools to support hypermedia linking for stored and streamed audio (the prototype tools were demonstrated at ACM Multimedia 1997).

In the next section we present the motivation for our work, and this is followed in Section 4 by a description of a content based retrieval tool using our alternative database design. Section 5 discusses our hypermedia linking tools and explains the use of contours in content-based navigation. The implementation is described in Section 6, with a discussion of future work in Section 7.

3. MOTIVATION

The ability to follow hypermedia links to audio files is a standard feature of hypermedia systems; 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 less common but nonetheless desirable. Furthermore, we envisage situations which benefit from links between parts of audio files. In traditional audio applications, the data is linear and unstructured; in contrast, our work regards audio as a branching, structured medium, like hypertext but with the additional challenge of working with streams as well as the store-and-forward model more readily adopted for multimedia document delivery. We are interested in the authoring, transport and delivery of this branching material.

With branching audio, different listeners can experience different tours through this structure, and they can interact to influence their route. In the case of speech radio this may 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, hearing an unfamiliar term, the user can ask all available links to be computed dynamically

and follow one to a glossary. This involves 'pushing' the programme to the user, with the possibility of the user 'pulling' additional information based on location or what has gone before.

This paper addresses branching audio with musical content. Traditionally, interaction with musical structure is an activity associated with the composition, production and publication process, rather than the listener. Hence the tools we describe here are aimed primarily at the 'authors' rather than the readers. However, our work also addresses scenarios where a user may navigate the musical structure, for example, to find a particular piece of music or to match the music to an activity they are pursuing. Readers are also authors; listeners are composers. Consider, for example, the evolution of portable audiocassette and compact disc players to wearable computers playing audio streams from the Internet.

Where does the structure come from? Some information, such as song structure, is available from existing production processes and might be discarded at present, but could be retained and enhanced. However, archive material lacks this information, so there is a need for tools which can determine structure. Whatever metadata might be available, there are always situations in which available links must be determined based on content rather than location. This provides additional motivation for content based retrieval and navigation of musical documents.

4. CONTENT BASED RETRIEVAL

Given a contour, the content based retrieval tool returns a list of songs which match that contour, ranked according to an appropriate metric ('contour based retrieval'); the metric depends on the user's requirements and the source, and therefore the quality, of the contour. In our system the contour is extracted from a fragment of music, which the user may have identified by pressing a button ('LINK' in Figure 1) whilst listening to the music, or via some other interface. We use MIDI as a standard representation in the system, therefore the fragment should either consist of MIDI events or should be in a form that can be converted to MIDI (in principle this could be other media, such as a musical score or the soundtrack of a music video). This section describes the contour representation and design of the contour database.

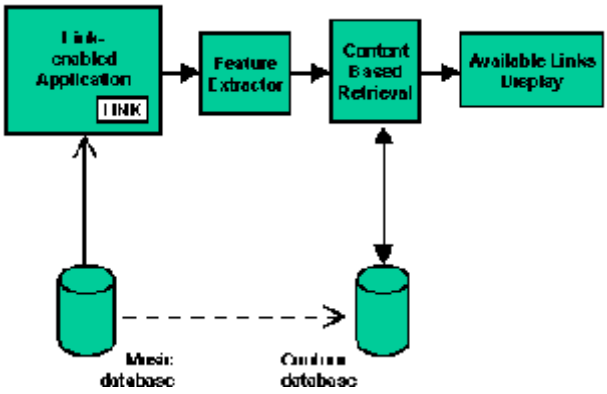


Figure 1: The content based retrieval architecture .

4.1 Contours

When considering how songs are recognised, the most obvious characteristic is that tunes are recognised independent of the key that they are sung in. To be independent of transposition, a database can cope with this by recording intervals (i.e. pitch changes) rather than absolute pitch. However, intervals may not be accurate, and research suggests that the *direction* of the interval is important [4], so music can be indexed and searched using just pitch direction.

A contour describes a series of relative pitch transitions, an abstraction of a sequence of notes. In the simple contour representation adopted for this work, a note in a piece of music is classified in one of three ways: it is either a repetition of the previous note (R); higher than previous note (U); or lower than the previous note (D). 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 fewer symbol than notes as only the transitions between notes are recorded. NB We use R (for *repeat*) in the same way as Ghias *et al* use S (for *same*), as we reserve S for another contour representation.

With respect to a query by humming (QBH) system, the use of contours eliminates input errors which are due to the user singing out of key, out of time or out of tune. As long as the pitch direction is correct then the contour should be found. The drawback is that all rhythmic information is lost; if this could be used in conjunction with the pitch contour then the number of incorrect matches would be decreased. We discuss refinements of this process, including other contour representations, in Section 7.

4.2 Deriving Contours

In our system we derive contours from sequences of MIDI events; in turn, we can derive sequences of MIDI events from digital audio files using pitch tracking. We are not solely interested in QBH: our queries may come from other sources, and as with QBH these queries are prone to error, though the types of error may be different.

MIDI files consist of a collection of tracks specifying for each instrument what to play and when to play it. It is not always possible to know in advance how useful a track will be when deriving contours; for example, it is probably not useful to store drum tracks. In the absence of reliable heuristics for

identifying prominent melodic features, we rely on an established standard for MIDI files called General MIDI, which matches instrument numbers with a known set of instruments (e.g. rhythm tracks must transmit on channel 10). Converting a MIDI track is a simple matter of identifying the pitch directions when a note is played. However, one channel may carry several notes sounding simultaneously, so it is necessary to adopt a policy such as taking the most recent note or, when several notes can be identified as commencing together (a chord) taking the lowest note.

Techniques for monophonic pitch tracking are widely documented (see, for example, references in [6]) and these enable contours to be determined from fragments of digital audio. Polyphonic pitch tracking requires more sophisticated solutions and is very sensitive to the style and instrumentation of the music. It is significant that for our application we do not require perfect polyphonic pitch tracking; in fact, monophonic tracking of prominent melodic features in polyphonic source material may suffice. We have developed a polyphonic pitch tracking algorithm which employs an iterative technique to optimize the analysis of spectral data, and this has delivered very promising results for certain classes of music.

4.3 The Database Structure

The systems described in [11] and [6] both have characteristics which are not well suited to our intended application in content based navigation (see Section 5). The former only indexes the first part of each piece, while we need to identify all alignments, and we need to improve on the scalability of the latter to ensure response times within bounds for an interactive CBN system.

The contour for a track is long, for example it averages 170 pitch directions per minute of our test material. The contour database stores *sub-contours* and uses them as the key on which to search the database; to store a whole pitch contour it is split into overlapping sub-contours of *key_length* pitch directions. For example, where *key_length* = 12, the contour

DURDRUURDRUDUR

is split into the overlapping contour set:

DURDRUURDRUD
URDRUURDRUDU
RDRUURDRUDUR

The length of the key is important as it defines the number of distinct contours that may be held in the database; more contours allows the database to be more accurate. The number of contours is given as:

$$\text{max\_contour} = \text{alphabet\_size}^{\text{key\_length}}$$

A contour can be one of 3 letters (U, D and R) so, for a key length of 3, the number of possible contours would be 27. This means that the system could only differentiate between 27 queries, which only need to be 3 pitch directions long. This is not enough to identify a piece of music; the *key_length* should be long enough to uniquely identify a song. The research by Ghias *et al* [6] found that 12 notes (not pitch directions) were enough to identify 90% of their 183 songs. To allow selection from a larger number of songs, the key length should be as high as possible.

4.4 Matching

To match a query, we produce a *near match set* of contours which are within a user defined distance of the query. The distance is a string metric which quantifies the minimum cost of transforming one string into another. Cost weights may be assigned to the individual editing operations involved in such transformations, namely symbol substitution, insertion, and deletion.

The metric depends on the source of the query, which might be extracted from a MIDI file or from a digital audio file with monophonic or polyphonic content, and it may come from original source material or via the tune recall skills of the user. By default we employ the Levenshtein distance 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 QBH, more research into tune recall is needed to identify suitable weights. It may also be possible to establish an appropriate distance metric automatically by learning from a reference set of data.

4.5 The Near Match Set

A brute force approach to collating the near match set would iterate sequentially through all contours. It is possible to implement a similar algorithm using a tree structure to visit each contour.

Consider a tertiary tree whose depth is *key_length*. 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 contain the set of all possible contours (see Figure 2).

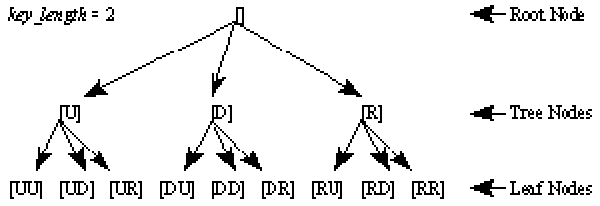


Figure 2: The contour tree.

A simple tree search would compare the contour at each leaf node with the query, as in the brute force approach; indeed,

it has the same time complexity as that approach.

The 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 the tree. This permits large sub-trees to be removed, reducing the number of leaf nodes. This vastly improves the performance.

There is a further improvement that can be made. Currently each node contour is compared. There are two situations where the distance value (*d*) 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. Take an example where there is one error in the contour and there are two levels of the tree still to be recursed. If three errors are allowed overall then the whole of the sub-tree must be within tolerance.

This means that if *d* = *key_length* then all leaf nodes could be added without comparing the contour at each node. This improves performance because the first *d* levels would not be compared. This decreases the time taken to match for large values of *d* but also has a noticeable effect for smaller values (ie: $d < 3$).

All contours stored in the database are *key_length* pitch directions long. If a query is submitted with more pitch directions than this, a more accurate search of the database is expected. This is achieved by building a list of sub-contours from the query, as used when adding a contour to the database (see Section 4.3). Approximate matching is performed on each sub-contour and the near match sets pooled. This larger set is then used to search the database.

4.6 Ranking of 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 less usual in a text retrieval system for the query to be so prone to error and in this respect contour matching is more akin to spell-checking. This means that the policy for ranking results is different, giving priority to a large number of hits over a single precise hit.

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

5. CONTENT BASED NAVIGATION

In this section we extend content based retrieval to content based navigation. We first discuss our hypermedia tools, and then the use of contours in this framework.

5.1 Open Hypermedia

The hypermedia model we have adopted to work with branching audio is an example of an *open hypermedia system* [3]. There are two key aspects to the model we have adopted:

1. Information about links between multimedia documents is stored separately from the documents themselves (in contrast to common practice with HTML documents on the Web). This information is stored in link databases (*linkbases*) and by selecting different linkbases, the user obtains different views of the documents. Access to linkbases may be abstracted into a *link service*.
2. When the user requests available links, the linkbases are queried with either the location in the stream or with a feature extracted from the data at that location. This means it is always possible to link from somewhere in a document. Linking from content in this way is an example of a *generic link* [8].

This model was employed in the Microcosm [5] open hypermedia system, which also provides an audio tool called the *Soundviewer* [7] supporting hypermedia links into and out of stored audio. The MAVIS system [9, 10] applies open hypermedia to other stored media, notably images but also including stored audio for proof of concept.

Our objective is to explore open hypermedia applied to audio, in particular streamed audio. Since the existing systems do not directly support streamed media or distributed working, we have implemented a component-based approach in order to support this experimental work. Hence our system resembles the Microcosm architecture but works with streamed media and contemporary link services, such as the *Distributed Link Service* [1]. The incorporation of musical objects in a hypermedia system, as both static and temporal media, has been discussed elsewhere; e.g. Ossenbruggen [12].

5.2 Navigation

At any place in a piece of music (e.g. whilst listening to the music), a user may ask for available links. Their position in the music, or the position of a selection they have made by whatever means, is the *source endpoint* of those links. This is compared with the link information in the link database to determine all the possible *destination endpoints*. The information in the link database, which essentially consists of stream and position information in this case, is generated by the authors of links.

In the absence of any other link information, CBR is a useful

tool to assist the user in navigating the information space. Although it performs a function which resembles link resolution, in that it generates available links from a given musical fragment, these are strictly links to similar material; this should be contrasted with the different types of available links that might be obtained from a link database. In fact, the CBR tool could be seen as a means of obtaining similar source endpoints rather than a mechanism for link resolution *per se*. For an author, creating a linkbase, it is another tool to assist in identifying endpoints.

In content based navigation, a contour extracted from the current position or selected fragment is used to interrogate the link database, thus obtaining a list of available destinations according to the content rather than the location of the source endpoint. Links with contours as their source endpoint are *generic*: the endpoint corresponds to any musical fragment that matches the contour, in any document.

The metric used to determine matches may be very different to the QBH case, depending on the quality of the feature extraction and the quality of the data in the linkbase. Linkbase data can be very high quality, having been 'hand-crafted' and checked by the author, and in some applications the user works with multiple, small linkbases rather than one large linkbase. In this situation, and with contours extracted directly from MIDI selections, we are able to employ a straightforward substring comparison. However, an error-prone contour (perhaps derived by polyphonic pitch tracking from the selection) still requires an approximate match, and in some situation linkbases can be very large. It is in these situations that the CBR techniques of the previous section can also be applied to CBN.

5.3 System Design

Figure 3 shows the architecture of the system for CBN. The Link Manager is the central application and is responsible for supervising link resolution; all the other tools communicate with it. For example, when a user selects LINK from a viewer application, information about the current position or selection in the audio stream is sent to the Link Manager, which then resolves the link by interacting with other components and generating a list of possible destinations. The Available Links display presents these destinations to the user, who can then select any of these in the usual way; if the *Auto Follow Links* option is enabled, a link will be followed as soon as it is added to the display. The viewer is any application that can be programmed to send link information to the Link Manager. We have created a viewer which is a simple wrapper application for the underlying multimedia system.

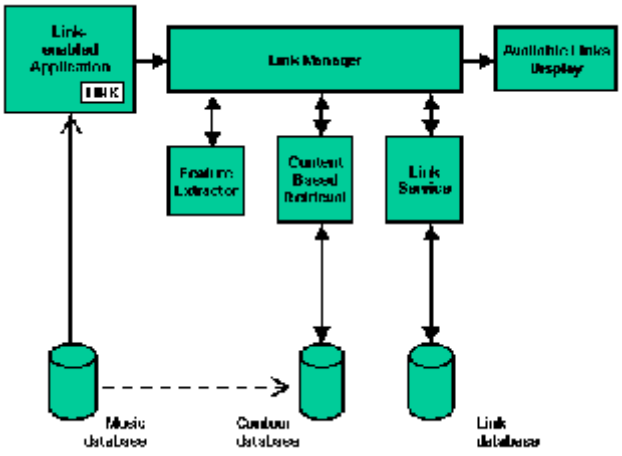


Figure 3: The content based navigation architecture.

The feature extractor uses information about a selection in an audio file to obtain the content of that selection and then extracts one or more contours (according to the number of tracks present). These contours are returned to the Link Manager, where they can be resolved independently or in combination. Given a selection in a digital audio file or a movie, the feature extractor can call the pitch tracker to generate the MIDI selection.

The Link Manager performs link resolution via a link service. At its simplest, this is a simple file lookup, or a database query. However, the link service may be a third party service (such as the Distributed Link Service), perhaps remote from the host, and the Link Manager can communicate with this via the appropriate protocol. A third party link service which supports generic links in text may also be able to deal with contours to an extent, but in general the link service needs to implement the appropriate contour matching. Hence the matching algorithms that we have adopted (e.g. Levenshtein and substring matching) need to be implemented in, or available to, the link service. In fact the system we describe could itself be regarded as a link service, consulting other link services in turn [2].

6. IMPLEMENTATION

6.1 Content Based Retrieval

The CBR tool implements the database structure described in

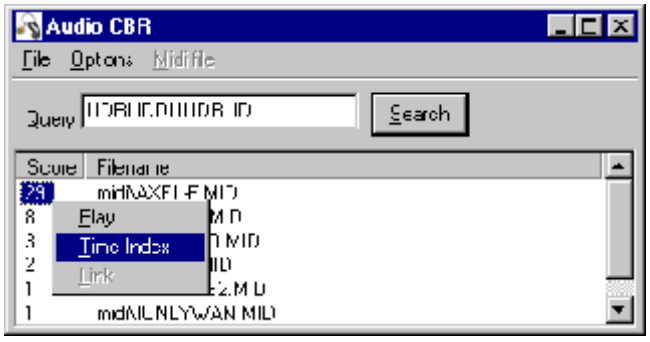


Figure 4: The content based retrieval tool.

Section 4, accepting a contour as a query and displaying matches with their scores (see Figure 4). Having selected one of the matches, the user can opt to play that song or to compute information on alignment of the contour matches; i.e. the positions at which the contour occurs, expressed as a time index suitable for use as an endpoint. Figure 5 shows the time indices for the best scoring match of the 12-symbol contour shown in the previous figure.

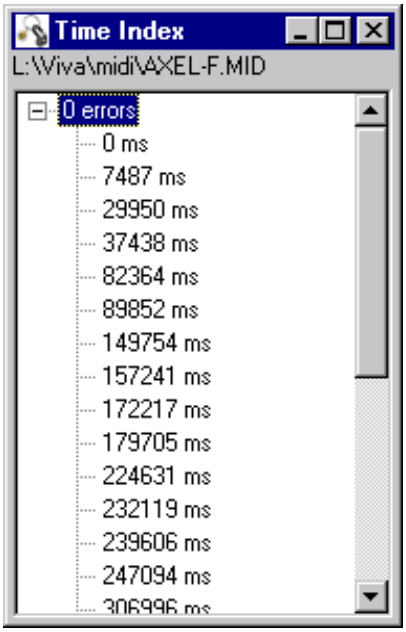


Figure 5: The time index display.

The system has been tested with a database of 8000 MIDI files, including some multiple versions and duplicates. The database was analyse to determine the practicality of contours. Figure 6 shows the number of files matched per contour. There are 531,441 possible contours which have twelve pitch directions. Of these, 60,000 contours each identify just two files. The graph continues beyond matching 60 files, some contours are contained in over 1000 MIDI files. This shows that the principle of using contours to identify music does work, although a larger *key_length* is required to improve performance and uniquely identify a file.

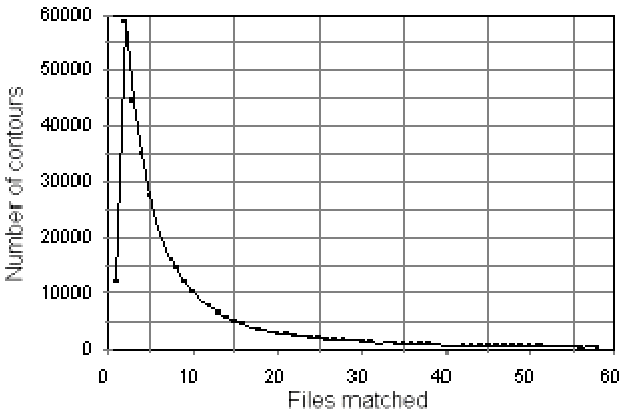


Figure 6: The number of files matched per contour.

The performance of the system is suitable for content based navigation. On a single user Pentium 133 system, exact matches take 0.01 seconds on average for a query with *key_length* pitch directions. Matches allowing for one error in the query take 0.12 seconds.

6.2 Audio Tools

The architecture described in Section 5 was implemented using Borland C++ Builder for Microsoft Windows NT v4. We took a component based approach to implementation as it allows straightforward expansion of the system and evaluation of different component implementations.

The central component is the Link Manager as it is the point of contact for all link-enabled applications. A link-enabled application can be anything which is capable of communicating with an OLE Automation server, such as a custom written program or many applications with a macro language. Figure 7 shows the Link Manager in diagnostic mode with the Available Links display (and source endpoint).

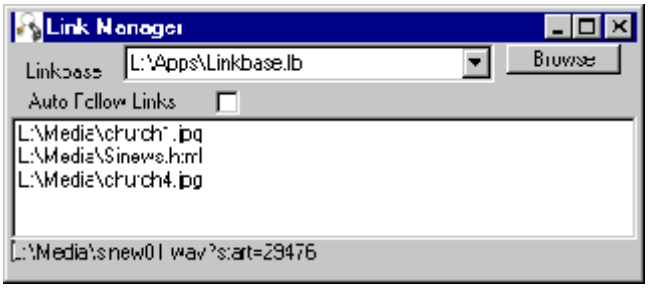


Figure 7: The link manager.

Our main viewer is the Link Player, which is a wrapper application for the MS Windows Media Control Interface (MCI) and looks very much like other media players; the main difference is the addition of a Link button, which sends the current position or selection information to the Link Manager (see Figure 8). In fact the player has several options which control when an endpoint is sent to the Link Manager: *Link on Play* sends an endpoint whenever the play button is pressed; *Auto Link* sends an endpoint at regular time intervals.

Figure 8: The link player.

Each Link Service is an OLE Automation server which implements a common set of functions. Each service takes an

endpoint and resolves it against a linkbase, returning a list of endpoints. We plan to use the close relationship between OLE Automation and DCOM to distribute the link services. The feature extraction is currently built into the Link Manager although we plan to use DLLs or OLE Automation to allow multiple feature extraction engines. Each engine takes an endpoint and returns a list of equivalent endpoints. The resultant endpoints may then be submitted to any of other engines until a complete list of endpoints is built.

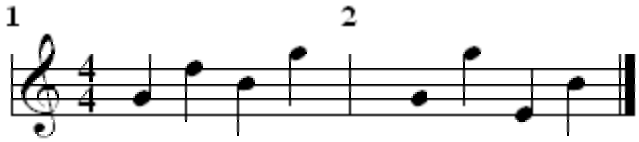
7. DISCUSSION AND FUTURE WORK

7.1 Contours

Our experience is that, while suitable for QBH, the contour representation is not sufficiently specific for some of our applications, in particular linking from selections in MIDI files when a precise hit can be expected. A lot of information is thrown away, which might otherwise help reduce the search space, such as rhythm information and the pitch intervals. We are investigating other representations, such as secondary contours. Given that we have an effective tool for working with contours based on an alphabet of three and that we are already matching with multiple contours, our approach is to work with multiple simple contours rather than developing a single, compound contour; these approaches may be equivalent.

One of these is the time contour. Similar to the way a pitch contour describes a series of relative pitch transitions, a time contour describes a series of relative note lengths (or the length of time between each note). Time in a piece of music is classified in one of three ways: it is either a repetition of the previous time (R); longer than the previous time (L); or shorter than the previous time (S). Thus, the piece can be converted into a string with a three letter alphabet (L, S, R). The duration over which a contour lasts may be used to further reduce the search space; for example, this helps avoid one bar of a melodic part matching another part that changes note just once per bar, such as a bass line.

Pitch contours are designed to be an abstraction of the notes to allow for errors in the input. This is useful for both hummed queries and the result of feature extraction engines. While pitch errors are likely to accumulate over time, more information can be deduced from notes which are close together. Using the example musical score below, the pitch contour for both bars is identical, "UDU", although they are clearly very different to listen to.



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 we observe that 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". This only applies when the pitch direction changes, otherwise all but the first pitch direction would be "up a lot". We propose a single character representation of the five pitch direction types based of: u, U, r, d and D for up, up a lot, repeat, down and down a lot respectively. The first bar of the example is represented as "udU", while the second bar is "uDu", showing a difference.

We have adopted an alternate approach 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. Although the possible secondary contours are constrained by a given primary contour, preliminary experiments suggest this approach to be very effective in reducing the search space.

7.2 Audio Tools

The tools have achieved their objective of providing an environment for investigating open hypermedia applied to audio and in particular, content based navigation. We support streaming via RTSP [13] and we see this as the means for transporting branching audio, hence we are exploring the use of RTSP for transporting endpoint information, linkbase information and for carrying queries. It has emerged that a key area for further work is the state of the session as we do not currently support a notion of 'back'. In fact, there is no concept of a *description* of a session, and we are therefore investigating session descriptions such as in SMIL [14].

Contours are just one feature which we can extract from musical content and we are working on others. We are particularly interested in a hybrid approach, combining the results from different analyses, and the architecture of the tools will evolve to support this. We are reviewing pitch tracking techniques, as our requirements are different to other applications.

The experience of applying open hypermedia principle to audio has been enlightening. The idea that link information should be stored separately is uncontroversial when there are no standard formats which embed it! In fact, we believe there are situations when it *is* useful to embed endpoint or link information, and this is readily achieved in audio file formats

without violating open principles, because they typically support the notion of multiple channels. For example, we can encode endpoint information as MIDI events on an unused MIDI channel, enabling us to store, transport and even edit it using standard tools.

ACKNOWLEDGEMENTS

We are grateful to Hugh Davis for supporting the project and providing advice on open hypermedia, to Thomas Cooke for his remarkably good polyphonic pitch tracking algorithm, to Neil Ridgway and Lee Oades for the additional audio tools that made the ACM Multimedia 97 demo, and to Paul Lewis for discussions about MAVIS. This work is partially supported by EPSRC grant GR/K73060.

REFERENCES

1. Les Carr, David DeRoure, Wendy Hall, and Gary Hill. The distributed link service: A tool for publishers, authors and readers. In Proceedings of the Fourth International World Wide Web Conference: The Web Revolution, Boston, Massachusetts, USA, December 1995.

2. David DeRoure, Les Carr, Wendy Hall, and Gary Hill. A distributed hypermedia link service. In Third International Workshop on Services in Distributed and Networked Environments, pages 156-161, Macau, June 1996. IEEE.

3. Hugh Davis, Wendy Hall, Ian Heath, Gary Hill, and Rob Wilkins. Towards an integrated information environment with open hypermedia systems. In Proceedings of the Fourth ACM Conference on Hypertext, Models for Open Systems, pages 181-190, 1992.

4. W.J. Dowling. Scale and contour: Two components of a theory of memory for melodies. Psychological Review, 1978.

5. Andrew M. Fountain, Wendy Hall, Ian Heath, and Hugh C. Davis. MICROCOSM: An open model for hypermedia with dynamic linking. In Proceedings of the ECHT'90 European Conference on Hypertext, Building Hypertext Applications, pages 298-311, 1990.

6. A.Ghias, J. Logan, D. Chamberlin, and B. C. Smith. Query by humming - musical information retrieval in an audio database. In Proceedings of ACM Multimedia 95, San Francisco, California, November 1995.

7. Stuart Goose and Wendy Hall. The development of a sound viewer for an open hypermedia system. The New Review of Hypermedia and Multimedia, 1:213-231, 1995.

8. Wendy Hall. Ending the tyranny of the button. IEEE Multimedia, 1(1):60-68, Spring 1994.

9. Paul H. Lewis, Hugh C. Davis, Steve R. Griffiths, Wendy Hall, and Rob J. Wilkins. Media-based navigation with generic links. In Proceedings of the 7th ACM Conference on Hypertext, pages 215-223, New York, 16-20 March 1996. ACM Press.

10. P. Lewis, H. Davis, M. Dobie, W. Hall, J. Kuan, and S. Perry. Content based navigation in multimedia information systems. In Proceedings of ACM Multimedia 96, pages 415-416, New York, NY, USA, November 1996. ACM Press.

11. Rodger J. McNab, Lloyd A. Smith, Ian H. Witten, Clare L. Henderson, and Sally Jo Cunningham. Towards the digital music library: Tune retrieval from acoustic input. In Proceedings of DL'96, 1996.

12. Jacco van Ossenbruggen and Anton Eliens. Music in time-based hypermedia. In Proceedings of the ECHT'94 European Conference on Hypermedia Technologies, Technical Briefings, pages 224-227, 1994.

13. H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). Technical Report MMUSIC WG Internet Draft, Internet Engineering Task Force, March 1997.

14. SMIL (Synchronized Multimedia Integration Language), W3C Technical Report, November 1997.