Melody recognition systems

Carl Ansley

October 28, 1994

Contents

Chapter 1

Introduction

A customer walks into a record store to buy a new single that they have just heard on the radio. The computer system in the record store has an index of singles, which can be referenced by code number, title, or artist. Unfortunately, the customer does not know the name of the song, or who the artist is; the only information available is the melody itself.

In this situation, it would be desirable to index the tunes in the music database by their melody. But this raises some questions. How is it possible to perform a *search* on a melody? And exactly how can the melodies be indexed in the first place?

What is required is a *melody recognition system*. Such a system should take a human voice (humming or singing a tune) and produce a ranked list of the most likely matches to this tune contained within a *melody database*.

1.1 Pattern recognition

Humans have a remarkable ability to quickly recognise and understand patterns in all types of sensory input. For us it is a trivial task to associate a particular sound, smell, or visual cue with some meaning. Computer scientists have for years been attempting to provide computers with a similar ability, to recognise patterns in the digital data fed to them. Although the task of programming a computer to *understand* its input is a very difficult problem, the recognition and association of fairly complex patterns within the data is now becoming practical. This is in part due to better algorithms, and in part due to significant increases in the processor power of modern computer systems.

The degree to which the application of pattern recognition techniques have been successful is dependent on the nature of the data and the type of pattern being searched for. For example, computers are particularly adept at finding a word in a document — as long as the word and the document are given to the computer in a digital, character-based format. Such a feature is commonly found in word processors and text editors. A more challenging situation is where the document is in bitmap form and the search word a digital sound sample. A solution to the latter problem is several orders of magnitude more complex to implement than the former.

There are several applications where considerable motivation exists to find a workable solution to the more complex recognition problems. Some common examples are

- Optical character recognition (OCR),
- Optical music recognition (OMR),
- Note recognition, and
- Speech recognition.

The melody recognition system that is the focus of this report makes direct and indirect use of the first three recognition systems listed above. An OMR system incorporating OCR [Bai] was used to scan in a melody database from the hymn book "Hymns for Today's Church", which became a testbed for the melody recognition system.

Note recognition is the preprocessor of a melody recognition system. It is responsible for converting digital samples of a human voice humming or singing into note form.

1.2 Melody recognition

The purpose of *melody recognition* is to automate the process of finding a reference to a particular melody. The idea is that a computer is given a sequence of notes, which it then must match to a corresponding entry in a melody database. Once a match has been found, what happens next is entirely application-dependent. The output may be as simple as printing out the name of the matching melody, or as complex as playing it while printing out the sheet music.

There are several issues to be considered when implementing a melody recognition system –

- How does the user enter the notes of the search key into the system?
- How will the system deal with inaccuracies in user input?

- Once the notes have been entered, what sort of technique should be employed to compare melodies?
- What methods can be used to create and index the melody database?
- Can the system deal with potential inaccuracies in the music database?

These are discussed in Chapters 2, 3, and 4.

1.3 Project aims and objectives

The aim for this project was to research, design, and provide the basis of an implementation for a *real-time* melody recognition system. The system should match music hummed or sung by a human with a particular piece in a melody database. The system should also allow for very large melody databases, where linear searching of the database would be too slow.

The design of the system consists of the following components:

- Digital sampling,
- Note recognition,
- Melody recognition, and
- Melody indexing.

They are linked together in a serial manner, the output of one component being the input to the next. The overall structure of the system is shown in Figure 1.1. Some of the system components have been well researched (such as note recognition [Kuh90],[Lan90],[SJ89],[Ric90]) and others have not (such as melody recognition). The main thrust of this report relates to *melody recognition* and how this effects the indexing of the *melody database.*

An important part of this is experimentation with various possible melody correlation functions. A good function should have the following three characteristics —

- accuracy (melodies that almost match should be given much higher correlation values than those that do not),
- speed (essential if the system is to exhibit real-time performance), and
- index-ability (easy to modify to allow indexing, so matching can be done in less-than-linear time with respect to the database size).

Most project experiments were performed on a Sun SPARCstation 10, using a melody recognition test program called `mrs`, which is written in C.

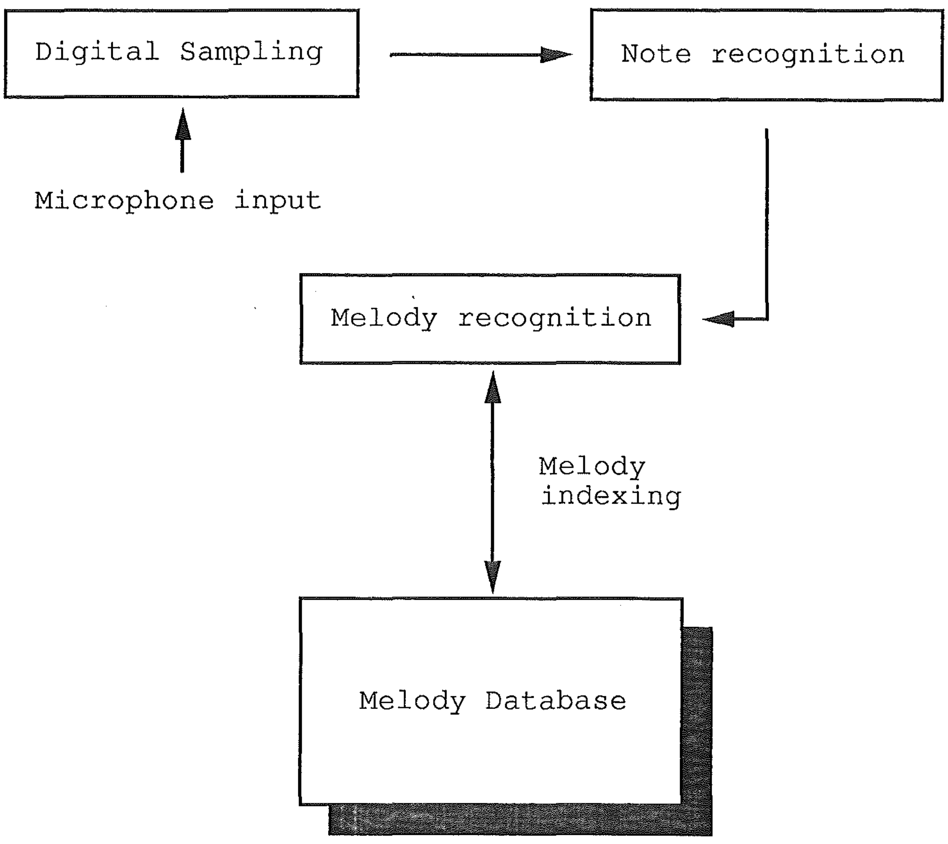

Figure 1.1: Melody recognition system diagram

1.4 Description of this report

Chapters 2 and 3 describe the issues relevant to note and melody recognition. Chapter 4 provides a background to the problem of indexing melodies. Chapter 5 describes the design and implementation of the `mrs` melody recognition program. Chapter 6 presents some results gathered through use of `mrs`. Chapter 7 contains some conclusions based on the experimental results.

Chapter 2

Note recognition

Note recognition is the initial phase of a melody recognition system. It is made up of the following components:

- Digital sampling of vocal input (Section 2.1),
- Note segmentation (Section 2.2),
- Pitch recognition (Section 2.3), and
- Pitch and rhythm quantisation (Section 2.4)

The efficiency and accuracy of these subsystems are very important. Although allowances are normally made for errors that occur during the note recognition phase, the overall performance of melody recognition system is highly dependent on the quality of this output.

2.1 Digital sampling

A sound can be described as a continuous function of time, where the dependent variable represents the *amplitude* (or loudness) of the sound signal at any given time. This is the *waveform* of the sound signal.

Because the waveform is continuous[1] it is impossible to represent it digitally with complete accuracy. This problem is overcome by approximating the waveform, in such a way that the most important information is preserved.

The first step is to convert the variations in air-pressure which make up the sound into variations in electrical voltage. This is normally performed by a

[1]Consequently, there are an infinite number of amplitude values between any two distinct time points.

microphone. At constant time intervals a sequence of measurements is then taken of the resulting voltage waveform. The frequency of these measurements is called the *sample rate.*

2.1.1 Amplitude quantisation

When a voltage measurement is taken at a particular time, the resulting value is one point on a continuous voltage range. As such, this value would require an infinite number of bits to represent in a digital binary form. An approximation is found by dividing the voltage range into N subranges. All the voltage values which fall within a particular subrange are assigned a value which represents that subrange. This is the final step in the process, and is called *amplitude quantisation.* The quantised value requires approximately $\log_2 N$ bits to represent digitally.

2.1.2 Error introduced by digital sampling

There are two types of errors which are introduced by this process. The first is called *aliasing*, and the second is called the *quantisation error.*

Aliasing occurs when a high frequency near the sample rate is mistakenly interpreted as a very low frequency. A common method of reducing the effect of aliasing error is to use an aliasing filter in conjunction with the analogue to digital converter.

The quantisation error is the difference between the original voltage level and the mid-point of the voltage range in which it lies. The amount of random white noise introduced into the sampled signal by the quantisation error is inversely proportional to the sample rate. In other words, a higher sample rate results in less noise.

2.1.3 Implementation

For this project, all sound sampling was performed on a Macintosh, using the built-in sound sampling software. The sample rate was 22,254 samples per second, each sample represented by an 8 bit value (giving 256 distinct voltage subranges). This is more than adequate for the purposes of the pitch recognition subsystem. The magnitude of quantisation and aliasing errors are not significant enough to degrade recognition performance.

2.2 Note segmentation

When a digital representation of the user's input has been obtained, it is necessary to split the sound sample into chunks. Each chunk should represent a single note. This process, called *note segmentation*, splits a sound sample at the points where the RMS[2] power of a signal drops below a certain threshold.

The threshold can either be static or dynamic, and for note recognition the latter is normally used. A static threshold assumes the user will sing at a relatively constant volume. A dynamic threshold has the advantage that it can adapt to varying levels of volume.

When the process of note segmentation is complete, the resulting chunks of sound sample can be individually passed to the pitch recognition system. The information calculated during this phase is also used to perform rhythm quantisation.

2.3 Pitch recognition

Two pitch recognisers were considered for use in this project. The first is based on an algorithm called the *Fundamental Period Measurement* [Kuh90].

The second method is based on the *Gold Rabiner* [GR69] algorithm. The code for this technique was obtained from Trevor Monk's **SightSinging Tutor** application [Mon93], and is written in C. The code also performs note segmentation, pitch quantisation, and rhythm quantisation.

2.3.1 Fundamental Period Measurement (FPM)

With this method, the sound samples are fed through a bank of six bandpass or lowpass filters whose upper cutoff frequencies are spaced at half octave intervals. If the filters are sufficiently sharp, then one of the filter outputs will be basically sinusoidal. The period of this sinewave and the pitch of the input signal can then be determined by measuring the time between zero crossings.

The algorithm used to make the final pitch decision is shown in Figure 2.1. The maximum of the amplitude measurements A(0)...A(5) is determined and compared with a silence threshold. If the input is silent, this is encoded as a period of -1. Otherwise the amplitude measurements are scanned, beginning at the lowest frequency filter, for the first filter with an appreciable amplitude.

The fundamental of the input signal is then assumed to lie in the passband of

[2]Root Mean Square

```
max_amplitude = max( A(0), A(1), ..., A(5) )
if max_amplitude < SILENCE_THRESHOLD then
   period = -1
else
   threshold = max_amplitude / 4
   period = -1
   filter = 0
   got_period = FALSE
   while filter <= 5 and got_period == FALSE
      if A(filter) > threshold
         if T(filter) is reasonable for filter then
            period = T(filter)
         elseif filter < 5 and T(filter+1) reasonable for filter+1 then
            period = T(filter+1)
         else
            period = -1
         endif
         got_period = TRUE
      endif
   endwhile
endif
```

Figure 2.1: Pseudo-code for the Fundamental Period Measurement algorithm

this filter or the filter immediately above it. The period in the filter's output is read and checked to see if it is reasonable for the filter. If so, it is taken as the period of the input signal. If not, the period of the next highest filter is read. If it is reasonable for its respective filter, then it is taken as the period. Otherwise, the period is set to -1 (equivalent to silence), representing a problem in recognizing the pitch.

This algorithm was implemented on an Acorn A5000 in C. The main advantage of the technique is that it is computationally efficient, primarily driven by Kuhn's original motivation of achieving real-time pitch recognition on an early IBM PC. The memory requirements of the system were negligible, and Kuhn's implementation used merely 50% of the available processing power of an IBM PC operating at 4.77 MHz.

2.3.2 Gold Rabiner algorithm

The Gold Rabiner algorithm operates on a single lowpass filtered waveform. The peak to peak and peak to valley measurements of the filtered waveform are fed into six parallel pitch period estimators. A coincidence detector is used in combining the six estimators into a final output pitch.

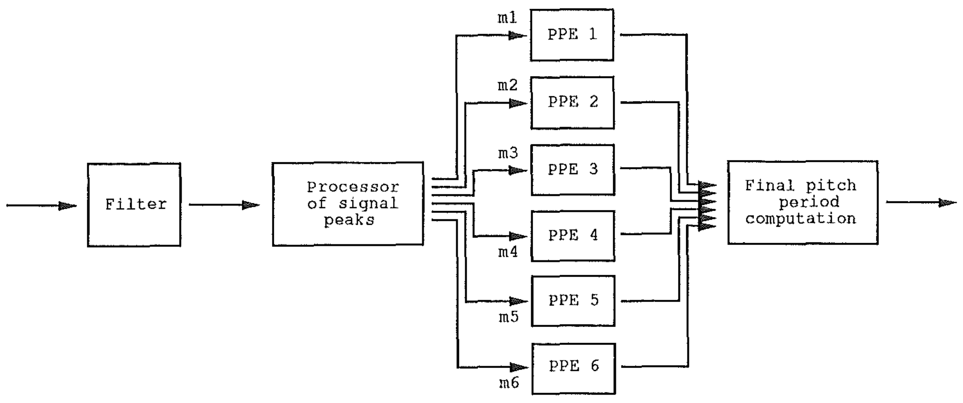

Figure 2.2: Block diagram of the Gold Rabiner algorithm

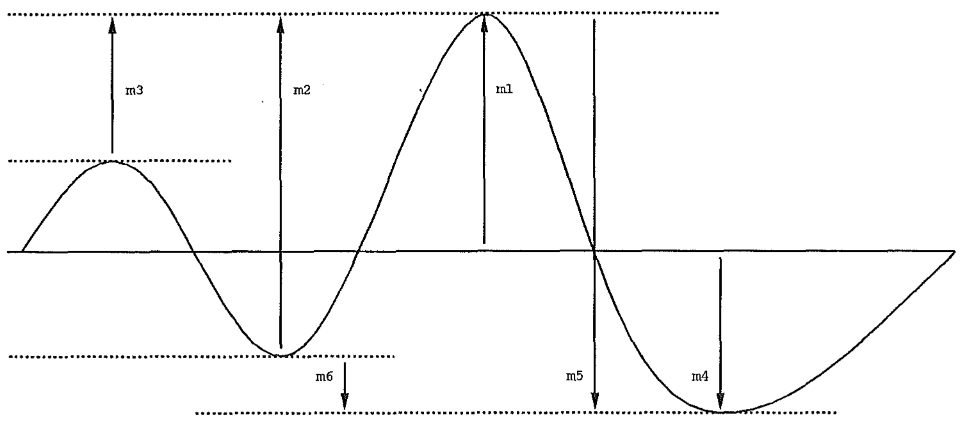

Figure 2.3: Estimator inputs generated from a portion of filtered waveform

Figure 2.2 shows a block diagram of the Gold Rabiner algorithm. The initial filter uses a lowpass cutoff of 900Hz, which ensures the fundamental frequency and at least two higher harmonics are present in the signal. The signal peak processor generates inputs for the six pitch period estimators (PPEs) by scanning the filtered waveform for local maxima and minima. Figure 2.3 shows an example of estimator inputs generated from a portion of filtered waveform. The waveform has two local minima and two local maxima.

Each estimator operates on its own pulse train, analysing the height and distance between pulses. They use this information to generate their own estimates for the final output pitch. The last stage of the algorithm is a majority estimate combiner. It takes the six pitch estimates produced by the PPEs in combination with previous estimates and determines the final output pitch.

Sample name	Sample size	FPM	GoldRabiner
JBOND	222540 bytes	1.2 seconds	9.2 seconds

Table 2.1: Comparison of two pitch recognition programs — FPM vs GoldRabiner

The code for this algorithm was originally written in C on a Commodore Amiga. The tasks of pitch and rhythm quantisation are integrated into the software. It was converted by Monk to work on the Apple Macintosh platform. For the purposes of this project, the code was converted to operate on an Acorn A5000 and subsequently modified to output pitch deltas. Section ?? describes the format of this information.

2.3.3 Implementation

Table 2.1 shows a time comparison between FPM and Gold Rabiner processing a 10 second sound sample. No note segmentation, rhythm quantisation or pitch quantisation was performed. The test machine was an Acorn A5000 with 4 megabytes of RAM and a 33Mhz ARM CPU without floating point co-processor.

The FPM implementation is significantly faster than the Gold Rabiner implementation. Much of the difference can be accounted for by the following factors:

- GoldRabiner uses emulated floating point instructions.
- FPM uses purely integer arithmetic.
- GoldRabiner is written entirely in C.
- FPM is written mostly in C, but the function to perform parallel waveform filtering is written in ARM assembler.

Both algorithms exhibit $O(N)$ time complexity. FPM has six filters operating in parallel, GoldRabiner requires only one. The six pitch estimators and complicated majority estimate combiner account for a majority of the execution time used by GoldRabiner. The remainder of the FPM algorithm is extremely simple and requires very little processing time. Because the six parallel filters are the most computationally intensive components of FPM, Kuhn (1990) regarded them as ideal candidates for implementation in hardware, or as a hardware/software hybrid. With the significant increase in processing power over recent years, this is unlikely to be necessary in future systems.

Despite the faster implementation of the FPM algorithm, the Gold Rabiner code was chosen for use in this project. The three reasons for this are (1) the implementation was entirely in C and therefore more portable, (2) the difference in speed is mainly implementation related, not algorithm related, and (3) the tasks of pitch and rhythm quantisation are integrated into the code.

2.4 Pitch/Rhythm Quantisation

The final step of note recognition is assigning musical values to the notes. This requires both pitch and rhythm quantisation.

Rhythm quantisation simply takes the length of a note calculated during the note segmentation process, and quantises the note to the nearest rhythmic value. This should take into account varying values for tempo, as durations in musical rhythm are relative to the tempo. `mrs` currently ignores rhythm, for reasons outlined in Chapter 5.

Pitch quantisation involves selecting a music pitch for a note given the varying frequency over the duration of that note. A range of frequencies map to a single pitch, and any note whose determined frequency lies within this range will be quantised to that pitch.

Determining the overall frequency of a note, given that frequency varies over the duration of the note, is not a trivial problem. A common difficulty caused by many pitch recognition algorithms is that they generate irregular spikes at the start of a note. Most of these algorithms require some time to settle on the correct pitch.

Because of these random artifacts, calculating the average frequency over the duration of a note is not a very accurate method of determining overall frequency. Monk's Gold Rabiner code uses a technique called steady state frequency analysis [Mon93]. This selectively analyses the steady state frequencies to provide a more accurate value.

Chapter 3

Melody recognition

The note recognition system will output a series of notes, which in some way need to be matched with known pieces of music within the music database. In other words *melody recognition* will need to be performed on the input data. The following points need to be considered when constructing a melody recogniser:

- The input is monophonic, therefore the music in the database should also be monophonic. This simplifies indexing and querying considerably.
- The search key will not be complete or precise. For example, the search key may not begin with the correct note or consist of the correct pitch changes, even though the overall *pitch contour* is very similar to the original.
- The system should be fast. It should be assumed that the music database is large, so a linear-search type approach to the recognition problem is not appropriate. For example, the Library of Congress (as of 1990) has a collection of 2 million musical pieces. Uncompressed, this would result in large amount of data even if represented monophonically.
- Because of the potential size of the database, the music must be stored and indexed as efficiently as possible.

3.1 Pitch contours

Most people can recognise a melody, whether it is a pop tune on the radio or a theme as it reappears throughout a symphony. When creating a melody recognition system for use by humans, an important question to consider is what features of melodies we use for memory, and how these features are processed to aid recognition. This in turn defines which elements of a melody need to be represented in the melody database.

Psychological studies of human memory have indicated that relative pitch is more important than rhythm for the recognition of melodies [SHO85] [DC86]. The suggestion is that *pitch contours* should be the primary data structure used to represent melodies for the purpose of melody recognition. This is intuitive in the sense that rhythm tends to contain less variation than pitch, so therefore less information.

Sloboda [SHO85] conducted a study on a musical *idiot savant*, who is capable of memorising large-scale pieces of piano music in three or four hearings. An idiot savant is a person with a low general IQ who have exceptional talent in one particular field. In the study it was found that the subject, in common with other observed cases in the literature [Min23][Rev25][Vis70], memorised material mostly in terms of tonal structures and relations.

Dewitt and Crowder [DC86] conducted a series of experiments to investigate the influence of contour and interval information. They recognised that two of the more general features of melodies are (1) contour (the binary pattern of ups and downs of pitch direction), and (2) interval information (the ordered sequence of pitch distances along a logarithmic scale of frequency between any two adjacent pitches in the melody). A primary question Dewitt and Crowder set out to answer was what information, be it contour, interval, or both, is useful for the recognition of melodies.

In the study, subjects rated pairs of melodies as similar or different on a five-point scale. Six conditions were defined by two delays (1 second and 30 seconds between hearing the two melodies) and three item types (target, related, and lure). In *target* pairs, the second melody retained the contour and interval information of the first melody, being an exact transposition to another key. In *related* pairs, only contour information was retained, and with the *lure* pairs neither contour nor interval information was retained.

The results indicated that contour information had a larger influence on recognition, especially after short delays. Interval information had a more significant effect after longer delays than it did after shorter delays, but was still the lesser influence. In the ideal melody recognition system, both pitch contour and interval information would be used to search for matching melodies. However pitch contour is the most important of the two, as this is the primary structure humans use to remember melodies.

3.2 Pitch contour matching

If melody recognition is to be performed using pitch contours, the problem of *pitch contour matching* needs to be addressed. There are three basic approaches to matching a pair of pitch contours:

- Regular expressions, and other exact pattern matching techniques,
- Approximate pattern matching (eg `agrep`), and
- Pitch segment matching.

Pitch segment matching, as used by `mrs`, will be described in chapter 5.

3.2.1 Regular expressions

Pitch contour matching could be performed using *regular expressions.* A regular expression is a search pattern format that contains facilities such as wild-card matching.

The main problem with regular expressions with respect to melody recognition is that it is an exact pattern matching technique. No errors are allowed for, so variations in the user's search key will result in poor recognition performance. Furthermore it is a linear searching method by nature, and therefore difficult to apply sub-linear indexing methods to it.

3.2.2 Approximate pattern matching

Approximate pattern matching, as performed by `agrep` [WM91][WM92], is much more suited to the problem of melody recognition than simple regular expressions. The algorithm used by `agrep` will find all occurrences of a pattern with at most k errors. An error is specified to be either an *insertion*, *deletion*, or *substitution.* This type of technique is known as *dynamic programming.*

An insertion error occurs when a single character needs to be inserted into an approximate occurrence of the pattern so that it exactly matches the search pattern. Deletion is the same, except that a deletion needs to be performed before an approximate occurance matches the search pattern. A substitution error is where one character must be substituted by another before the occurance matches the search pattern. *agrep* also allows a weighting to be given to each type of error. For example, a deletion can be made to count as two errors instead of one.

Dynamic programming is potentially useful for melody recognition. Because of its approximate nature, allowances can be made for variations in the user's search key. Unfortunately it shares a problem with the regular expression technique, in that it is a fundamentally linear searching method.

3.3 Previous approaches to melody recognition and comparison

Very few references to existing melody recognition systems were found in the academic literature. A reference to a prototype Japanese melody recogniser was observed on the BBC science and technology television series *Tomorrow's World*. Unfortunately this was a commercial system, and no further information could be obtained. Two published recognition systems are described below.

3.3.1 `ntt` (Name That Tune)

The `ntt` program was written by Michael Hawley [Haw90] at the MIT Media Laboratory. The input is in the form of a melody played at a keyboard, transmitted to the computer via a MIDI interface (According to Hawley, at this point the input pitches are quantised to remove embellishments, but he does not specifically state how this is done). It then finds possible matches from a database of musical themes.

In the database, each melody is represented as a string of relative pitch changes, plus a starting tone. e.g. `[3,-1,4,2,C]` These pitch changes are coded as ASCII characters, the numbers representing an offset from ASCII '`0`'. This ensures that all the melodies are simple strings which can be sorted. Also, since the encoding is relative, transpositions will be found.

When searching for a match in the database, `ntt` simply needs to perform a binary search. However, there are some problems with this approach, especially with respect to untrained vocal input:

- No "fuzzy" approximate searches are done.
- For an input to match a tune in the database, they must both start at the same point.
- Precise pitch changes are unlikely when the input is vocal.

Because of the above points, an algorithm such as the one used by `ntt` is also not appropriate for providing a metric of "similarity" between two or more tunes. It can do little more than perform an action such as "Find a tune that starts like this in the database".

3.3.2 Minimum Message Length comparison of musical sequences

In his thesis, Cook [Coo94] applies Minimum Message Length (MML) encoding to the comparison of melodies. The roots of MML encoding are in information theory and the field of data compression. Simply put, MML says that if significant compression can be achieved by expressing one sequence in terms of another, then those two sequences must be related.

As far as the basic MML technique is concerned, matching is a binary operation. In other words, individual pitches either match or do not match, and transpositions of the same musical theme are not recognised as being related. In order to compensate for this, Cook suggests a further adjustment to the basic algorithm, called *interval matching*, which uses a probabilistic order-zero model to quickly adjust to constant differences between pitches.

Cook describes the major advantage of MML as its *objectivity*, with parameters determined by data rather than a pre-determined heuristic method. However, the MML technique is designed for music *analysis*, and is unsuitable for melody recognition. A major disadvantage in this respect is the unsuitability of MML for fast comparisons and indexing, because it still requires a comparison of every entry in the database.

Chapter 4

Melody indexing

The concept of indexing tunes by their melodic components has been around for several decades. The thematic index is one of the more interesting special indexes in the field of music. Unfortunately, a problem with most thematic index reference books is that the task of searching for a particular melody is not easy unless the searcher happens to have some specific musical knowledge. However, they are worth investigating for some intriguing ideas which have possible application in the field of computer-based melodic indexing.

4.1 Thematic indexes

Barlow's *Dictionary of Musical Themes* [BM48] was published in 1949, and contained an index of over 10,000 tunes. To use the dictionary to locate a particular melody, the theme must first be transposed to C major, and then the letter representation of the first few opening notes looked up in an alphabetic index. Table 4.1 shows a segment of Barlow's notation index.

C C B C A F	H666
C C B C A G	A61
C C B C B A	C232
C C B C B C B C	M512
C C B C B C D	M846

Table 4.1: Segment of the notation index from Barlow's *Dictionary of Musical Themes*

The letter and number to the right of each definition in the notation index indicates the place in the alphabetic section of the book where the theme may be found. The themes are listed in their original key with the name of the composition and the composer. Figure 4.1 includes one of the theme entries which is indexed in Table 4.1.

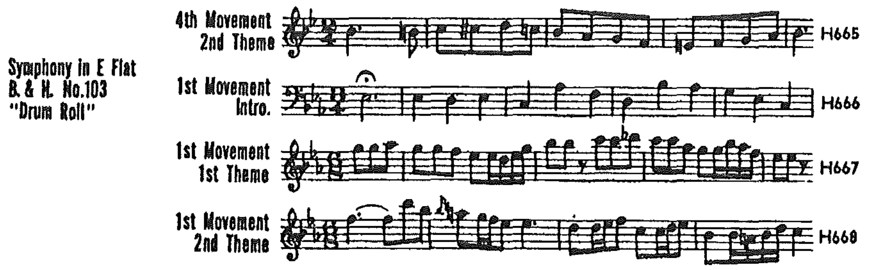

Figure 4.1: An example of theme entries from Barlow's *Dictionary of Musical Themes*

Most thematic indexes are devoted to the works of one composer. The *Melodic Index to the Works of Johann Sebastian Bach* [Pay62], for example, indexes and tabulates all of the themes of Bach according to their melodic design. The index contains 3872 themes, and enables the searcher to locate a Bach composition by the pattern of the first three intervals. All the melodies are in the treble clef and in the key of C. Repeated notes are ignored.

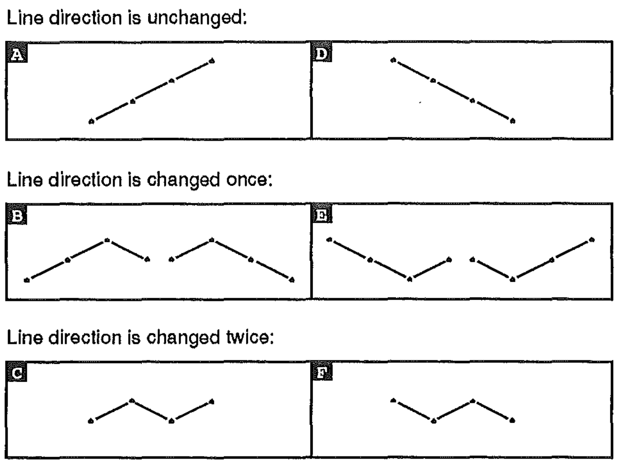
Line direction is unchanged:
A
D
Line direction is changed once:
B
E
Line direction is changed twice:
C
F

Figure 4.2: Six categories of melodic pattern used by the *Melodic Index to the Works of Johann Sebastian Bach*

The line formed by the composition's first four notes of differing pitch creates the pattern used in finding the composition. The patterns are categorised into six different types, as shown in Figure 4.2. Each of the six types are divided into seven sub-categories, based on the seven possible starting notes.

The thematic index, which is in the same order as the charts, lists the themes in full. Figure 4.3 shows an example of a finding chart, and Figure 4.4 shows the corresponding entry in the thematic index.

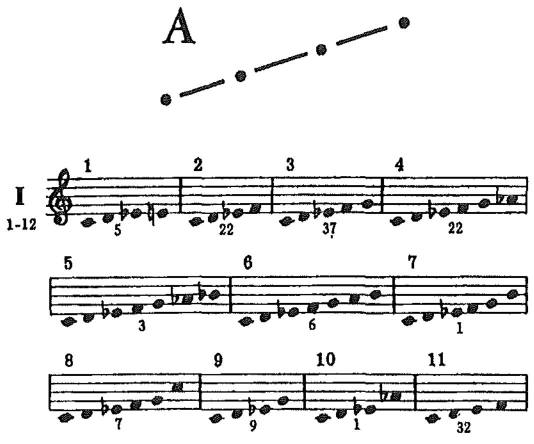

Figure 4.3: Example of a finding chart from the *Melodic Index to the Works of Johann Sebastian Bach*

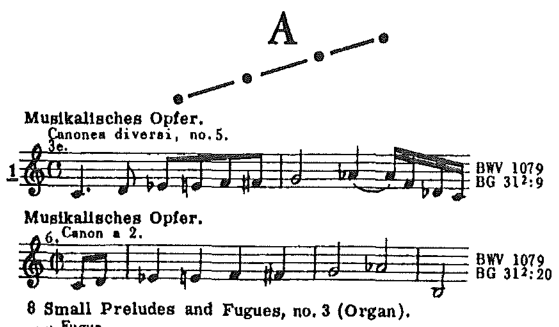

Figure 4.4: Corresponding entry in the thematic index

Chapter 5

An approximate melody recogniser – mrs

The program `mrs` was written as a testbed for measuring melody recognition performance using pitch contours. These contours are translated into a series of data structures called *pitch segments*, which can then be correlated using a variety of methods.

Due to the highly distributed nature of the entire system, it could not be effectively tested on real-time basis. However human voice search keys were entered into the system, and data was transferred from component to component manually. The system performed very well in the complete test cases that were done.

5.1 Construction of the melody database

The source of the melodies used in the construction of the melody database was the hymn book "Hymns for Today's Church". The melodies (approximately 600 in all) were scanned in one page at a time. Each page was processed by David Bainbridge's optical music recognition system. Figure 5.1 shows part of a page from the hymn book. Figure 5.2 shows the corresponding output after processing by the optical music recognition system.

The first section of the file specifies notes that make up each melody (there is one melody per stave). The second section of the file contains the text which appears immediately above or below each stave on the page. This text is treated as the "title" of the melody by `mrs`.

All files created by the optical music recognition system (one file per page of music) were concatenated and piped into `mrs`. At this stage `mrs` creates the

Delta value	Description of mapping
0	Much lower in pitch
1	Slightly lower in pitch
2	Slightly higher in pitch
3	Much higher in pitch

Table 5.1: Basic mapping of four-level pitch deltas

melody database, converting each tune into *pitch delta* format, described in Section 5.2.

The melody database is a simple text file, where each line represents a melody. A line contains the melody name followed by a tab, a colon, a space, and a corresponding pitch contour. An excerpt from the Hymn database is shown in Figure 5.3.

5.2 Pitch delta representation

To allow for errors in the input, the pitch change information generated in the note recognition sub-system is further quantised into *pitch deltas*. The unit of a pitch change is the semitone. All pitch changes of magnitude zero are ignored. This compresses the pitch contour horizontally to a significant extent, but has some advantages. The most important advantage is that it avoids ambiguities where the search key contains, for example, a single crotchet where two quavers of the same pitch was meant. Another advantage (albeit minor) is that it allows the number of delta levels to be a power of 2. This results in a convenient binary representation for each pitch delta.

`mrs` can handle either two-level or four-level pitch deltas. A two-level pitch delta can have only two possible values, **higher** or **lower**. Generating this is simply a matter of checking the sign of the pitch change.

Four-level pitch deltas are more complicated to create from the original pitch change information. The "fuzzy" mappings of the four possible values are shown in Table 5.1.

To perform the quantisation, the size and position of the four sub-ranges in terms of semitones need to be specified. Figure 5.4 shows the distribution of pitch changes present in the Hymns database.

Due to errors introduced by the optical music recognition system[1], some spurious pitch changes are present in the data. A pitch change of more than an

[1]Mainly due to the poor quality of the original sheet music.

Delta value	ASCII representation	Subrange (semitones)
0	`D`	$PitchChange < -2$
1	`d`	$-2 \leq PitchChange < 0$
2	`u`	$0 < PitchChange \leq 2$
3	`U`	$PitchChange > 2$

Table 5.2: Mapping of four-level pitch delta values to semitone subranges

octave (12 semitones) would normally not appear in a melody. However, these can safely be ignored as the number of glitches is too small to affect the basic distribution significantly.

Approximately half the pitch changes in the database have a magnitude of one or two semitones. Therefore, `mrs` maps half the available delta levels to changes of this magnitude. This ensures that roughly equal numbers of each pitch delta value will appear in the melody database. The actual mappings of sub-ranges to pitch delta values are shown in Table 5.2.

Also shown in the table are the ASCII representations used for each delta value. This representation is the one used by `mrs` to store melodies in the database.

Figure 5.5 shows an example of the overall process. The notes which make up the search key are converted into pitch changes, and then pitch deltas. Both the numeric delta values and ASCII representations are shown.

5.3 Pitch segments

Before pitch contour correlation can take place the contours are broken up into *pitch segments.* This conversion must be performed on both the search key and the melody database. Figure 5.6 shows two segmented pitch contours, one with a segment size of two and the other with a segment size of four.

In the case of the search key, several segmented pitch contours are generated. The encoding of each successive segmented contour starts on a different note in the original search key.

The number generated is equal to the segment size, as each contour represents a different possible correlation point. This removes the problem which would otherwise occur if an occurance of the search key did not start on a segment boundary within the matching melody. The system is also more resilient to deletion and insertion errors within the search key as a result.

There is also a second representation of tunes in the melody database where the number of pitch deltas is not a multiple of the segment size. Because all partially empty pitch segments are discarded, information would normally be lost at the end of the pitch contour. In the second representation, the end of the last segment is aligned with the last pitch delta in the contour. This ensures the entire pitch contour is represented in pitch segment form.

5.4 Pitch segment correlation

Pitch segment correlation involves comparing two melodies in segmented pitch contour form and producing a correlation value. Similar pairs of melodies should be given higher values than dissimilar ones.

Two pitch segment correlation methods are supported by `mrs` directly, one of which is a refinement of the other. Their purpose is to show the basic viability of segmented pitch contours as a technique for melody recognition.

The first correlation method is a very simple technique called `commonsegs`. It simply adds up the number of segments that a pair of pitch contours have in common, and uses this as the correlation value.

The second method is a refinement of `commonsegs` called `csordered`. This technique differs in that common segments in a pair of pitch contours are given extra weighting if they are in the same order. The overall correlation value is incremented for each pair of common segments that are in the correct order.

The results of the first method, `commonsegs`, were similar in nature to `csordered` except generally worse. The `commonsegs` method is weighted heavily in favour of longer melodies. Because ordering and melody length are not taken into account, longer tunes tend to be given higher correlation values simply because they contain more pitch segments. For the generation of experimental results `commonsegs` was ignored, and `csordered` was used exclusively.

5.5 Full-melody retrieval with `mg`

Given a search key and a melody database, a simple way of performing melody recognition is to obtain a set of correlation values for the search key and each melody in the database. Ranking the output in order of highest to lowest correlation value groups the most likely matching melodies for a given search key at the top of the list. This method was used to produce some of the results given in Chapter 6.

Although this proved successful for experimentation purposes, the technique is not appropriate for use in the melody recognition process. The main problem is that execution time is proportional to the length of the melody database. To reduce the time requirements of recognition, indexing of the database is required.

5.5.1 The mg system

The mg[2] system [WMB94] is a full-text retrieval system that runs under Unix. mg is designed to index and retrieve information from very large collections of documents.

The document collection is compressed and indexed before it can be queried. A wide variety of queries are handled by mg, but most importantly it can perform ranked searches without searching the entire database. This makes it an ideal candidate for use with mrs.

An inverted file is used by mg to perform indexing. Each item in the index is a "word" (in this case a pitch segment) followed by a list of "documents" (melodies) that contain the word. This means that only the melodies which have segments in common with the search key need be processed, the rest can be ignored.

5.5.2 mg and mrs

The mg system requires the document collection to be in a particular format before it can be compressed and indexed using the `mgbuild` program. Firstly, the `mg_get` shell script requires a slight modification so that it knows how to handle the melody collection. Figure 5.7 shows the lines that need to be inserted into the main `switch` statement.

This ensures that when the command `mgbuild melody` is executed, the output file of mrs (in this case `MelodyDatabase.mgseg`) is passed directly to the mg system without modification.

Secondly, mrs must be told to produce a slightly different format of melody database to the one described in Section 5.1. During the conversion of the optical music recognition output into a melody database, some extra operations need to be performed. A `Ctrl-B` character is inserted between every line in the database, which causes mg to treat the melodies as separate documents. The pitch contour of the melodies are segmented (as described in Section 5.3) by inserting a space between each pitch segment. This ensures that pitch segments are treated as separate words for the purposes of a ranked query.

Different letters must be used when outputting the ASCII representation of a 4 level pitch delta. "D", "d", "U", and "u" are used during the normal conversion process. However, mg's ranked query is case insensitive so the distinction

[2]Managing Gigabytes

between “D” and “d” is lost. Selecting four letters such as “G”, “d”, “u” and “B” solves the problem.

5.5.3 Melody querying

When the melody database has been compressed and indexed by `mg`, it can finally be queried. `mrs` can generate a query for a given search key to be subsequently processed by the `mgquery` program. For example, given the pitch contour `duduGBGBdu`, the output of `mrs` will be:

```
.set query ranked
 dudu GBGB  uduG BGBd  duGB GBdu  uGBG
```

In addition, `mrs` can create a very large query which performs a ranked search for every melody in the database. Executing this query on the Hymn database generated over 8 megabytes of output. This feature was implemented solely for experimentation purposes. See Chapter 6 for details of the results.

5.6 Summary of mrs command line arguments

```
usage: mrs <command> [command paramters]

<command> can be one of the following:  commonsegs, csordered, stats,
meltoseg, mgstats, meltomgseg, genmgquery, melpdelta

These commands take the following additional parameters:
  commonsegs <pitch contour>
  csordered  <pitch contour>
  stats      <filename>
  genmgquery <pitch contour>

stdin and stdout are used for input and output by all commands.
```

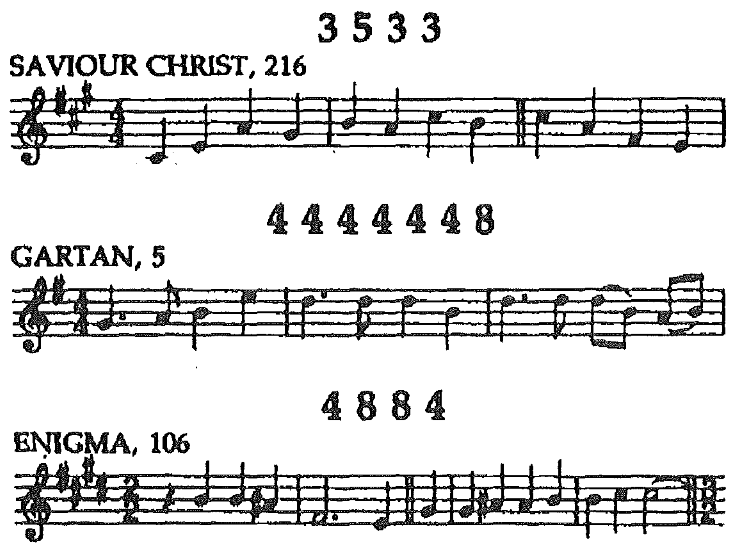

Figure 5.1: Page from original hymn book

```
1.1:  |  |  |
4F#(1.0),4C#(1.0),4E(1.0),4A(1.0),4G#(1.0) | 4B(1.0),4A(1.0),
5C#(1.0),4B(1.0) | 5C#(1.0),4A(1.0),4F#(1.0),4E(1.0) |

2.1:  |  |  |
4G(1.5),4A(0.5),4B(1.0),5E(1.0) | 5D(1.5),5D(0.5),5D(1.0),4B(1.0) |
5D(1.5),5D(0.5),5D(0.5)\_,_/4B(0.5),4A(0.5),4B(0.5)&4F#(4.0) |

3.1:  |  |  |  |  |  |
rest(0.5),4B(0.5),4B(0.5),4A(0.5) | 4E(0.5) | 4G(0.5) | 4G(0.5),
4A(0.5),4A(0.5),4B(0.5) | 4B(0.5),5C#(0.5) |  |

Stave 1:
Above: 3 5 3 3
      SAV OUR CHRIST 216

Stave 2:
Above: 4 4 4 4 4 4 8
      GARTA 5

Stave 3:
Above: 4 8 8 4
      u IGuA 106
```

Figure 5.2: Output from optical music recognition system

```
Saviour Christ, 216       : uUdudududdd
Gartan, 5                 : uuuUdduddu
Enigma, 106               : ddduuuu
The Infant King, 92       : uduuuuuudddduduudu
Harrow Weald, 2           : uudduudud dudddUd
Victor's Crown, 185       : uddUUDdUuduu
Ardwick, 224              : ddddUdudu
Passfield, 382            : DUduduUddDUuu
Schonster Herr Jesu, 209  : uduuuduuUd
```

Figure 5.3: Excerpt from the Hymn database

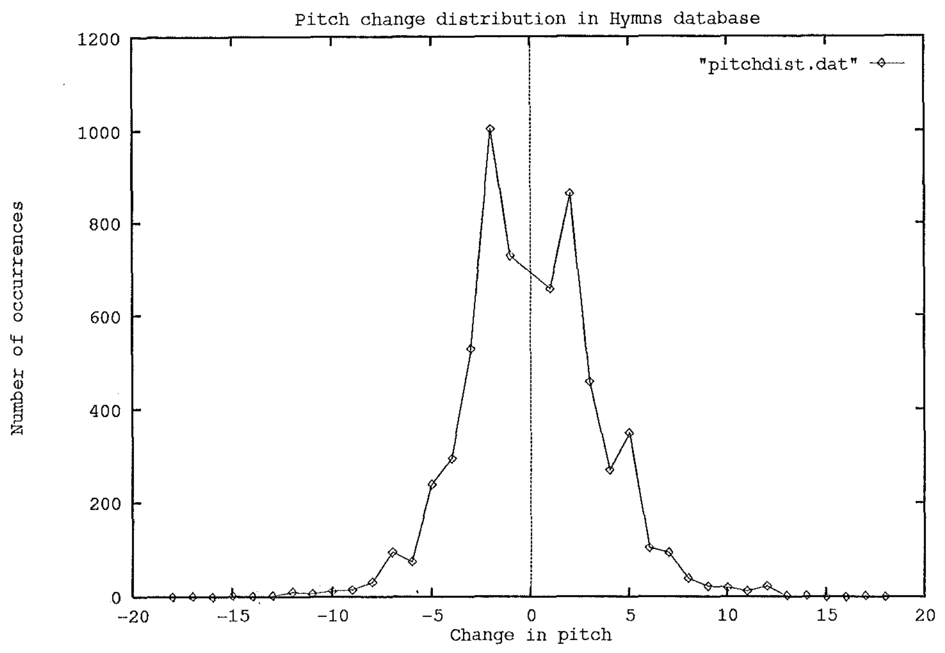

Figure 5.4: Distribution of pitch changes in Hymns database

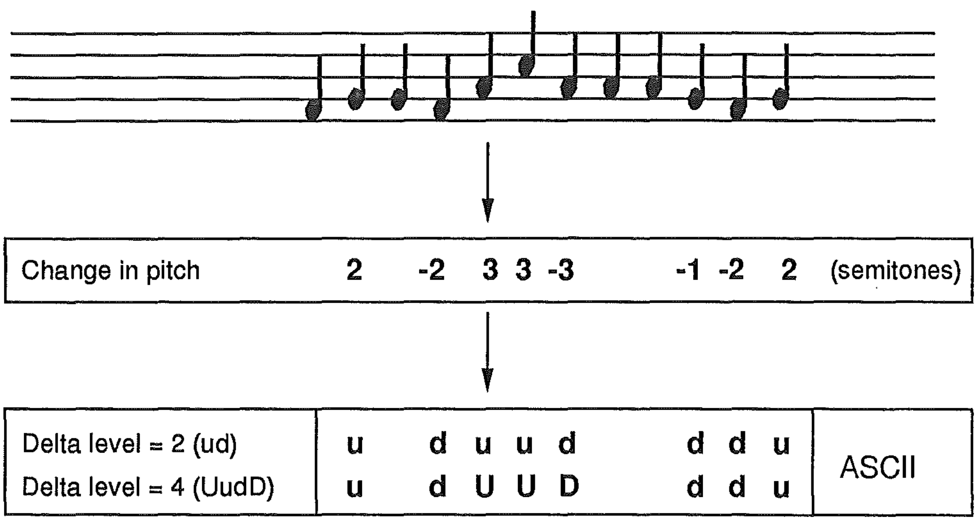

Figure 5.5: Conversion of notes to pitch deltas

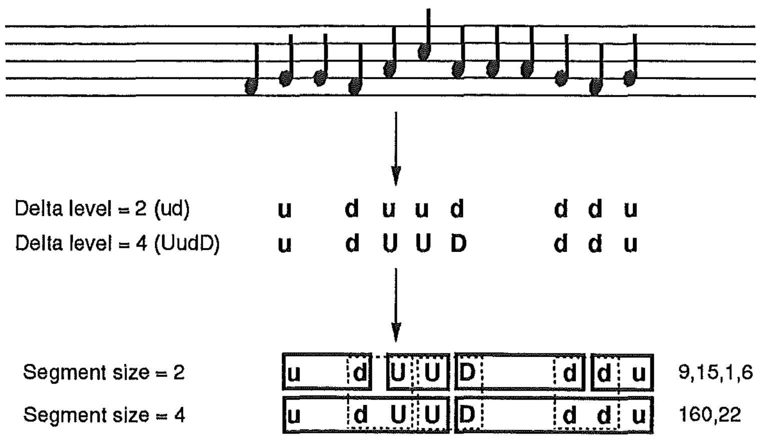

Figure 5.6: Segmentation of a pitch contour

```
case melody:
switch ($flag)
  case '-init':
  breaksw

  case '-text'
  cat MelodyDatabase.mgseg #insert database name here
  breaksw

  case '-cleanup':
  breaksw #-cleanup
endsw #flag
breaksw #melody
```

Figure 5.7: Lines that need to be added to the main `switch` statement of `mg_get`

Chapter 6

Experimental results

The chapter presents the results gathered from a number of experiments. Data was obtained on various combinations of pitch delta levels and segment sizes, to examine the effectiveness of pitch segments as a data structure. The performance and practicality of using mg as a method of indexing the melody database is also examined.

6.1 Pitch delta level and segment size

The results presented in this section were obtained in the following way. Given a search key and the Hymn melody database, a set of correlation values were generated for each possible pairing of search key and melody in the database. The output is ranked in order of highest to lowest correlation value. Therefore the most likely matching melodies for a given search key are put near the top of the list, with the least likely at the bottom.

However, the output is not a list of melody names or pitch contours. It is a descending list of numbers, where a number at a particular position in the list represents the correlation value of the melody at that position.

This list is generated many times, one for every melody in the database. In each case a different melody from the database is used as a search key. This means that the search key will always be a "perfect match" for itself. The results in Table 6.1 and Figures 6.1 and 6.2 were obtained by averaging the values of each position for all the lists.

Pitch delta levels	Segment size	Match position
2	2	95.25
2	3	44.46
2	4	19.08
2	5	9.23
2	6	4.61
2	7	2.03
2	8	1.05
4	2	6.04
4	3	1.15
4	4	0.53

Table 6.1: The performance of various combinations of pitch delta levels and segment sizes

6.1.1 Average match position

The *average match position* is the mean position of the perfect match to search keys in the set of output lists. A match position of 0 means that the melody recogniser has found the correct value. Therefore the performance of the melody recogniser is closely related to how small the average match position is.

Table 6.1 compares the average match position obtained using various combinations of pitch delta levels and segment sizes. A pitch delta level of 2 and segment size of 2 performed very badly, with an average match position of 95.25. The best performance was obtained with a pitch delta level of 4 and segment size of 4. The average match position in this case was 0.53.

6.1.2 Distribution of correlation values

Figure 6.1 shows the distribution of correlation values using 2 level pitch deltas. Figure 6.2 shows the equivalent graph for 4 level pitch deltas. Again, the combination of pitch delta level = 4 and segment size = 4 was the best performer, exhibiting the most "L" shaped curve and lowest correlation values for non-matching melodies.

There are a number of reasons for the poorer melody recognition performance observed with few pitch delta levels and small segment sizes.

A delta level of two, combined with a segment size of two, results in only four different segments. Therefore any reasonably long melody is bound to have copious quantities of all possible segments. This leads to a fairly flat distribution of correlation values, and an almost random expected match position.

A problem with the csordered technique in general, but one which is exacerbated by fewer pitch delta levels and smaller segment sizes, is that of *rank distinction*. Several of the most likely candidates for a match tend to be given the same correlation value. The main reason for this is that the csordered technique does not take into account melody length. If shorter melodies were given higher weightings when calculating the correlation value, this problem would be greatly reduced.

6.2 Indexing with mg

The experiments in this section differ from the previous section in that the correlation values were computed in cooperation with mg's ranked query facility.

Figure 6.3 shows the distribution of correlation values using 4 level pitch deltas and a segment size of 4.

The distribution of the graph is similar to its csordered counterpart. However, this technique exhibited two desirable properties:

- The average match position was exactly zero. The melody recogniser found the correct value every time, compared with 0.53 for csordered.
- The problem of rank distribution disappeared. The correlation values output by mg were floating point and had a much higher granularity to the corresponding values generated by csordered.

6.3 Approximate melody recognition

Figure 6.4 shows a *normalised* distribution of correlation values using 4 level pitch deltas and a segment size of 4. This test was slightly different to the previous mg test in that an error of one randomly-placed deletion was introduced to each search key.

By comparing the graphs of Figure 6.3 and 6.4 it is clear the introduced error had only a marginal effect on recognition performance.

6.4 Summary

Using csordered, the best results were obtained with -

- pitch delta level = 4, and

- segment size = 3 or 4.

Using `mg` to index the melody database proved very successful. The problem of rank distribution disappeared, and the system selected the correct match for a particular search key every time.

The system performed very well even when a random deletion error was introduced to the search keys. Recognition performance was affected only slightly.

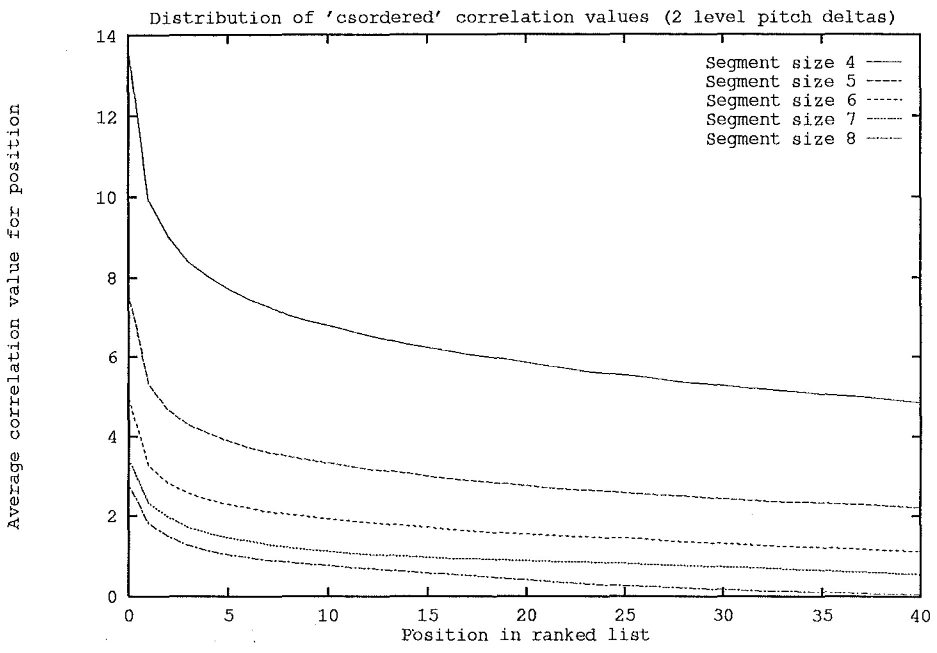

Figure 6.1: Distribution of correlation values with 2 level pitch deltas

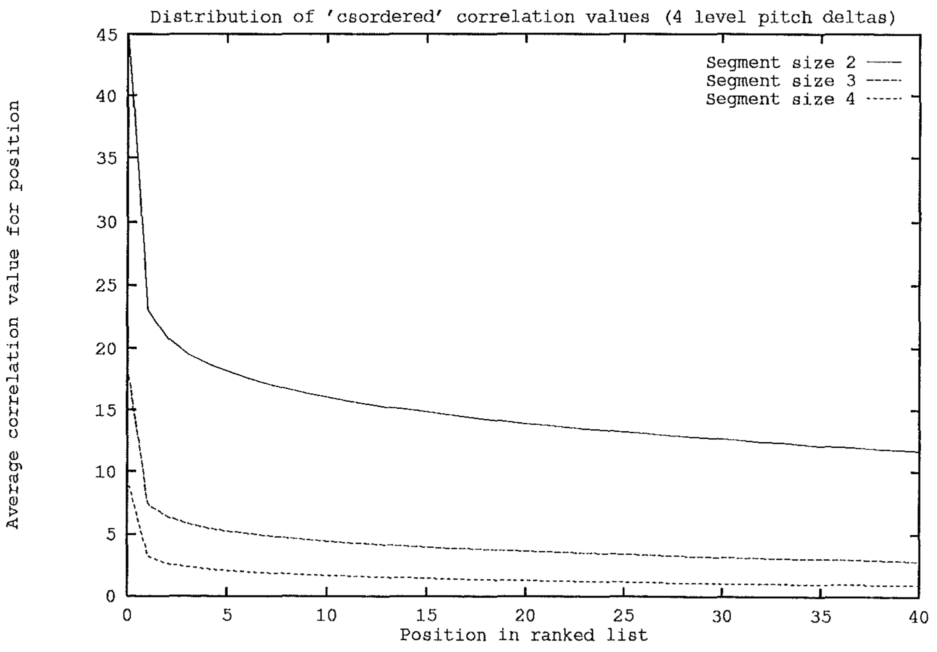

Figure 6.2: Distribution of correlation values with 4 level pitch deltas

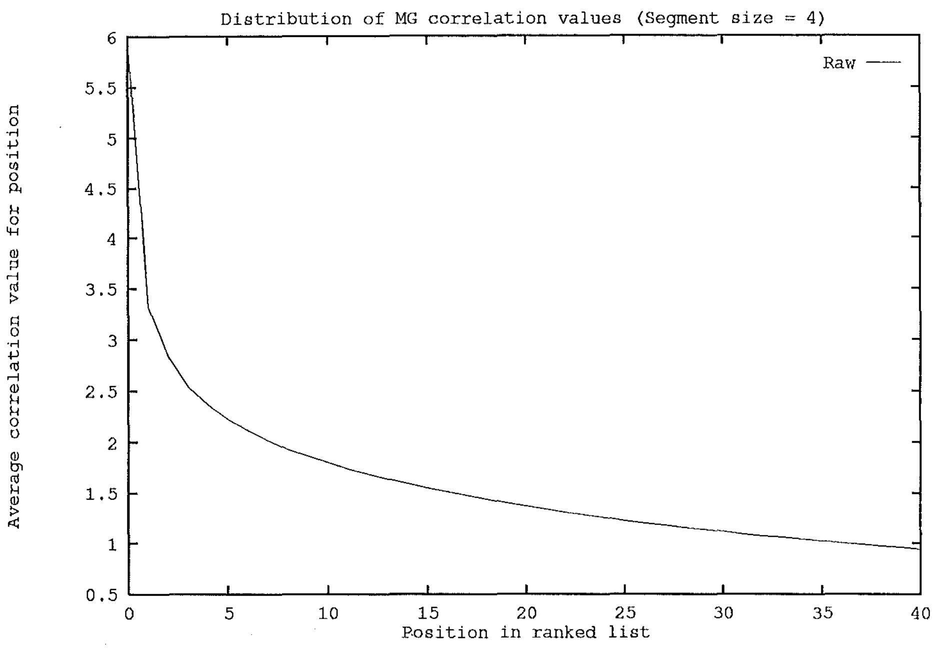

Figure 6.3: Distribution of correlation values

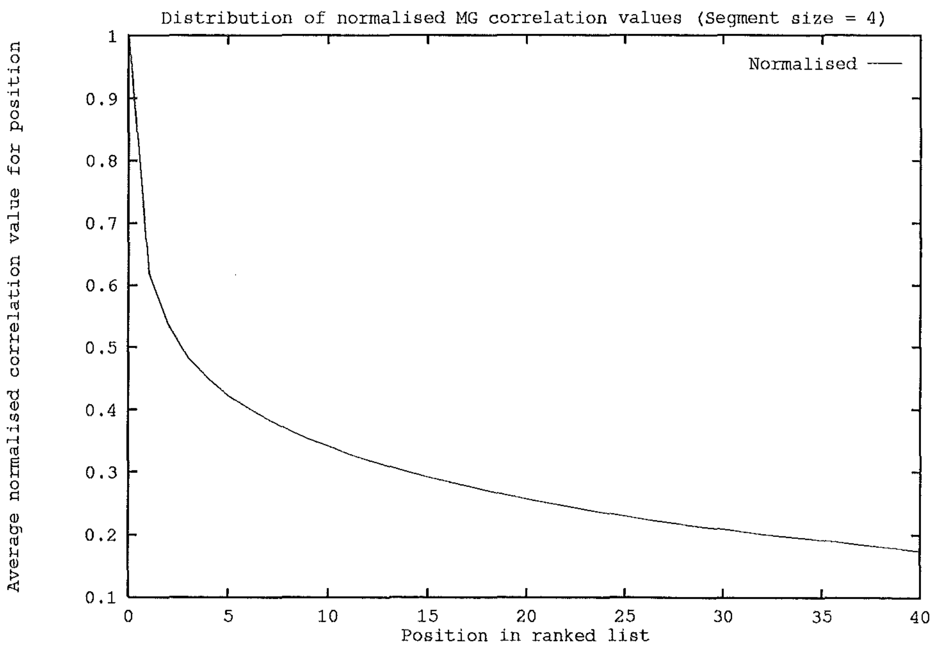

Figure 6.4: Distribution of correlation values, normalised with deletion errors in input

Chapter 7

Conclusion

The aim of this project was to research, design, and provide the basis of an implementation for a *real-time* melody recognition system. The human brain recognises complicated polyphonic music with ease. However, melody recognition is a complicated task for a computer to perform. It combines a number of difficult problems (pitch recognition, note recognition and approximate pattern matching) into one system.

With respect to human melody recognition, the two most important features are recognised as being contour and interval information. Psychological studies of human memory have indicated that relative pitch is more important than rhythm for the recognition of melodies. The suggestion is therefore that *pitch contours* should be the primary data structure used to represent melodies for the purpose of melody recognition.

The concept of *pitch segments* was introduced as a potentially fast and accurate way of performing melody recognition using pitch contours. This allowed the application of a full-text retrieval system (`mg`) to the indexing of the melody database, and proved very successful.

Bibliography

[Bai] David Bainbridge. An optical music recognition system. This is a work in progress (Computer Science Department, University of Canterbury, New Zealand).

[BM48] Harold Barlow and Sam Morgenstern. *A Dictionary of Musical Themes.* Crown Publishers, New York, NY, 1948.

[Coo94] Shane Samuel Cook. Minimum message length comparison of musical sequences. Master's thesis, University of Waikato, 1994.

[DC86] Lucinda Dewitt and Robert Crowder. Recognition of novel melodies after brief delays. *Music Perception*, 3(3):259–274, Spring 1986.

[GR69] B. Gold and L. Rabiner. Parallel processing techniques for estimating pitch periods of speech in the time domain. *Journel of the Acoustical Society of America*, 46(2):442–448, 1969.

[Haw90] Michael Hawley. The personal orchestra, or audio data compression. *Computing Systems*, 3(2):289–329, Spring 1990.

[Kuh90] William B. Kuhn. A real-time pitch recognition algorithm for music applications. *Computer Music Journal*, 14(3):60–71, Fall 1990.

[Lan90] John E. Lane. Pitch detection using a tunable iir filter. *Computer Music Journal*, 14(3):46–59, Fall 1990.

[Min23] B.M. Minogue. A case of secondary mental deficiency with musical talent. *Journal of Applied Psychology*, 7:379–352, 1923.

[Mon93] Trevor Monk. Computer assisted sightsinging training: Implementation of a sightsinging tutor (sst-1). A thesis submitted in partial fulfilment of the requirements for the degree of bachelor of computing and mathematical sciences, University of Waikato, Hamilton, New Zealand, 1993.

[Pay62] M.D. Payne. *Melodic Index to the Works of Johann Sebastian Bach.* Schiermer, 1962.

[Rev25] G. Revesz. *The psychology of a musical prodigy.* 1925.

[Ric90] D.M. Richard. Godel tune: formal models in music recognition systems. In *ICMC Glasgow 1990, Proceedings*, pages 338–340, Glasgow, UK, 1990. ICMC.

[SHO85] John Sloboda, B. Hermelin, and N. O'Connor. An exceptional musical memory. *Music Perception*, 3(2):155–170, Winter 1985.

[SJ89] Hajime Sano and B. Keith Jenkins. A neural network model for pitch perception. *Computer Music Journal*, 13(3):41–48, Fall 1989.

[Vis70] D.S. Viscott. A musical idiot savant: a psychodynamic study and some speculations on the creative process. *Psychiatry*, 33(4):494–515, 1970.

[WM91] Sun Wu and Udi Manber. Fast text searching with errors. Technical Report 91-11, Department of Computer Science, University of Arizonza, 1991.

[WM92] Sun Wu and Udi Manber. Agrep – a fast approximate pattern searching tool. In *USENIX Conference*, January 1992.

[WMB94] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes.* Van Nostrand Reinhold, 1994.