Music Information Retrieval: A Survey of Issues and Approaches

Ryan J. Demopoulos and Michael J. Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada
N6A 5B7
E-Mail: ryan@demopoulos.org, katchab@csd.uwo.ca

Technical Report #677
Department of Computer Science
The University of Western Ontario
January 2007

~ TABLE OF CONTENTS ~

1.0 Introduction

Music information retrieval, herein referred to as MIR, covers a broad range of topics not unlike those involved with the retrieval of text. The ultimate goal of MIR is to identify various words and properties contained within musical selections, storing these data within a musical database, and providing a query mechanism to specify searches over the database to retrieve the musical documents represented therein. Past research concerning textual databases has led to very accurate and efficient retrieval mechanisms; unfortunately, much work remains to be done in order for the retrieval of music to rival the characteristics of its textual counterparts.

Using computers for the purpose of retrieving musical information has been pondered since 1967, when Lincoln outlined criteria for a retrieval system that could index musical themes [Lin67]. From that time until the mid 1990's, casual research had been done to further refine the idea of eliminating the human component of audio transcription and searching; the bulk of this work actually targeted the speech transcription domain, with some potential findings applicable to topics in MIR [Foo99, KNKT95]. Recently over the last 10 years the scientific community has experienced an explosion of interest in the subject. Several factors have contributed to this. First, the emergence of better MIDI standards has led many to re-consider the use of the MIDI file format as a basis for music information sharing. MIDI is a clearly-defined mathematically-based musical file format, and thus simplifies musical analysis and categorization for database storage in a number of ways. Other factors, such as the increase of freely and publicly available repositories of downloadable music via the internet, have increased the potential utility of musical databases by making the information available for searching. Additionally, compression formats such as Ogg Vorbis, and more significantly MP3, have reduced musical file sizes to the point where a large number of songs can be reasonably stored within a single database; furthermore, this is becoming more and more feasible as the per-byte cost of data storage media continues to improve[1]. Finally, the recent interest in music information retrieval was partially sparked by a widely-cited and ground-breaking paper by Ghias et al. [GLCS95], outlined a framework for an online music information retrieval system with appropriate query mechanisms. The insight provided by this paper gave reason to believe that music databases would not only be useful to have, but clearly possible to achieve.

While MIR research encompasses a number of different approaches, the most common of these tends to borrow from common practices in string matching. There exists many application domains that already make use of string matching, most notably molecular biology and topics in evolutionary tracking [CIMR+02]. However, string matching within a music information retrieval context is arguably more complex than matching for molecular biology; the latter is normally concerned with pattern matching within strands of DNA, which is derived from a significantly-reduced

[1] http://www.littletechshoppe.com/ns1625/winchest.html

alphabet and is easily extracted from real-world samples. Conversely, musical domains suffer from problems not common to biological domains; multiple kinds of information must be coordinated (such as rhythm, pitch), and this information is quite difficult to extract from musical waveforms and even more clearly-defined representations of music, such as MIDI.

This literature survey paper is organized as follows. The remainder of Section 1 provides background information on MIR topics, and attempts to identify the potential usefulness of the field. Section 2 discusses the various ways to form and store musical data, with discussion on the utility of each approach. Section 3 describes the complex area of transcription, exploring how data can be algorithmically transformed from common musical source forms to a more malleable format, such as one of those listed in Section 2. Section 4 gives insight into the various ways that musical information can be captured from a user to be formed as a query for MIR systems. Section 5 covers many topics relating to the identification and searching of musical patterns, and provides some detailed examples of how existing algorithms operate on musical data. Finally, Section 6 offers a summary of topics covered, and discussion regarding future areas of work.

1.1 Aspects and Challenges

The process of music information retrieval can be broken down into a number of finite and relatively disjoint sub-problems, which is beneficial from a research perspective, as individuals can tackle the entire domain using a modular or incremental approach. While each MIR system has it own approach, Figure 1 outlines the general flow of tasks required in order to achieve a fully-functional MIR system.

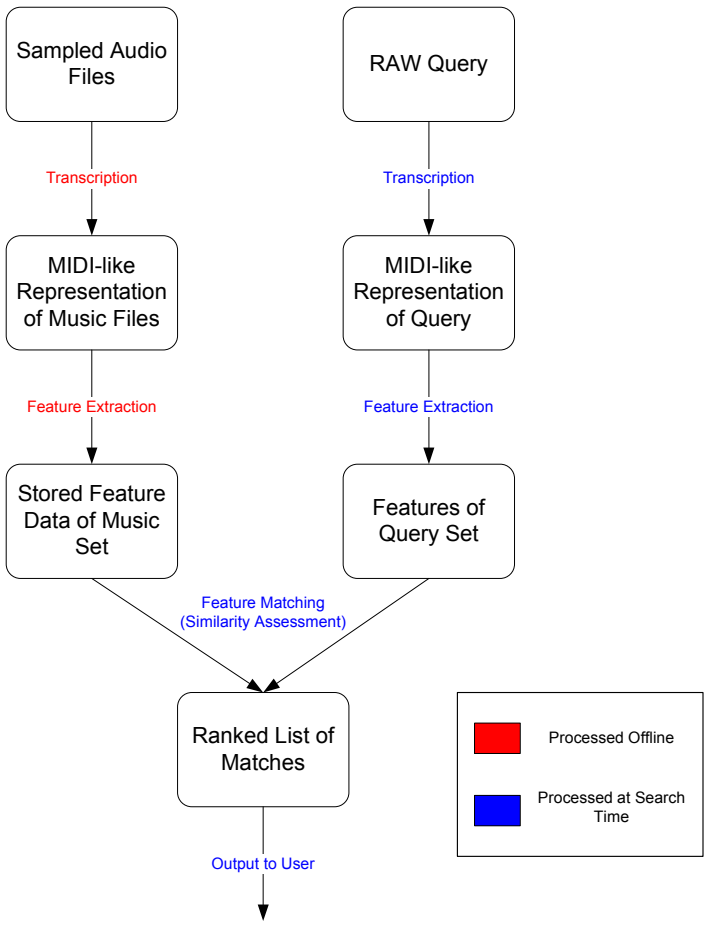

Figure 1
General flow of MIR system tasks, adapted from [Par05]

The steps outlined in Figure 1 cover several important aspects common to all MIR systems:

- **Transcription** – Normally, musical pieces or queries originate from forms that are not suited for standard matching algorithms. For example, suppose that a user sings into a microphone in order to find the piece that resembles their query. Their voice data will normally be captured as sampled audio, comprised of a waveform plotted against time. While this data format is excellent for providing an accurate reproduction of the user's voice, it is not appropriate for use in matching algorithms targeting a concrete musical representation such as MIDI. Thus, a reduction or *transcription* must take place, that will attempt to convert the user's query into a discrete sequence of pitch/rhythm values that are easily comparable to a target feature set. Section 3.0 covers the transcription process in depth.
- **Feature Extraction** – A musical feature can be defined as some aspect of a musical piece, phrase, or note such as pitch, rhythm, or an augmentation [DO06]. How can the pertinent features of music be efficiently and accurately identified? The approach taken often closely relates to the query method to be

used within the system. For example, if searching will take place by tapping a rhythm, then data will be categorized according to the rhythmic features of the various pieces contained with the database, and algorithms will be developed that attempt to accomplish this quickly. The ultimate goal of feature extraction is to reduce the data required to represent a piece or query, in order to identify matches across a very large database more efficient. By far, the most common features extracted take the form of patterns or repetitions in music, which are further investigated in Section 5.0.

- **Similarity Assessment** – This stage attempts to match a user's query to a subset (or feature) of a song within the database. The purpose of matching features rather than entire musical sequences (such as a whole query against all the musical information of each song in the database) is to decrease the workload of the matching algorithm. Consider that one piece of sampled music contains thousands of musical points to be matched, and thus can benefit greatly from a surface reduction. A variety of approaches exist that attempt to increase the accuracy and efficiency of feature matching, many of which are explored in Section 5.3.
- **Output to User** – How will results be ranked, and how can users quickly identify these results to ensure that they have achieved a correct match? While users could verify results by listening to the playback of an entire piece, it may take considerable time to evaluate a large result set. Thus, how can this process be streamlined for quick piece identification/verification?

Aside from the general tasks to be carried out as defined in Figure 1, there are other considerations that should be addressed by an MIR system. Two of these are:

- **Data Storage** – Two approaches exist in regard to data storage of musical information within a database. The naïve approach (not represented in Figure 1) is to store only the entire musical piece in the database, and each subsequent query over the data will re-analyze the various musical features of each piece in turn. This approach leads to significant inefficiency, as each piece is re-analyzed during each query. Alternatively, as demonstrated in Figure 1, music can be paired with additional feature information, extracted offline, that describes the piece itself. Queries thus will target these features, rather than the entire piece structure, to achieve faster response times.
- **Query Mechanism** – How will users specify a query? Most users of a music retrieval system will not have the skill required to specify a query using standard musical (score) notation, unlike textual searching where users are expected to understand how to form a query in a manner that closely resembles how the target data will be stored within the database. These and other considerations are explored in Section 4.0.

1.2 Real-World Applications

The realization of an efficient, fully-featured and fully-functional MIR system could dramatically impact the daily lives of many people. The music industry is driven by a large and diverse consortium of artists, producing countless songs and records; from a simple perspective, being able to search music could be a key component of leisurely activities and products such as karaoke machines, interactive MIR systems at record stores, or a resurgence of the jukebox with searchable titles. One can imagine the addition of musical searching to today's common home theatre systems or portable audio devices, such as iPods[2] and cellular phones.

Many individuals whose lifestyles and/or careers rely upon music would be significantly impacted by the availability of MIR systems over the internet. For example, librarians would be able to assist students in finding musical pieces available for loan, and vocal instructors could find sheet music of songs for their students. Also, live DJs and musical choreographers could make use of such a system to locate tracks that meet requests for a particular style of music [DR01]. In addition to these, MIR systems could be useful for the composition of new music; for example, songs with a certain *sound* could be identified and fed into learning algorithms of an automatic music generation engine, and real human composers could search with their own newly-composed pieces as queries to ensure that their work does not infringe on another composer's copyright. The potential for real-world use is ever-increasing as newer technologies surrounding the music industry continue to emerge, and this in part provides encouragement for researchers attempting to solve MIR-related problems in the various areas explored by this paper.

[2] http://www.apple.com/itunes/

2.0 Musical Representation

Most approaches to the searching component of MIR require that musical data be organized in a clear and simple manner that specifically targets some algorithm to maximize efficiency. Consider the example of searching for a number in an ordered list of integers. Carrying out such a search will require O(n) time complexity if the integers are stored in a linked-list data structure, and yet this time can be reduced if the integers are ordered within an array. Likewise, if an MIR system attempts to match musical records based on sequences of pitch *and* sequences of rhythm, then it is optimal to first consider how that data can be organized to be efficiently handled by a searching algorithm.

Many of the algorithms explored in this paper attempt to format data before performing operations on it; thus, two clear families of musical representation can be defined. Comprehensive representations are those that present music in its "natural" form, or in a manner which largely captures and describes the data of the music as it was originally played and recorded. Reduced representations are almost always the result of some reduction carried out on a comprehensive representation, usually for the purpose of searching a particular feature of the music for data retrieval. Within a musical database, the process of extracting a feature (feature extraction) from a full musical surface for storage within the database can be referred to as document indexing, or sometimes pattern extraction as well. Further information on this topic can be seen in the sections immediately following, as well as Sections 3.0 and 5.0.

2.1 Comprehensive Representations

The following sections outline different ways that music can be stored/specified while covering many musical features, typically in order to allow the music to be reproduced accurately.

2.1.1 Sampled Audio Representation

The most common representation of music available on the internet is sampled audio[3, 4, 5]. This data is achieved by taking thousands of samples of sound every second, and then storing the resulting information in a potentially large file resulting in huge quantities of data on the musical piece. Because each sample carries a frequency and amplitude of sound, a smooth waveform is generated that can very accurately reproduce the music as it was originally played.

[3] http://en.wikipedia.org/wiki/Wav#Popularity
[4] http://en.wikipedia.org/wiki/Mp3#Internet
[5] http://findarticles.com/p/articles/mi_m0FOX/is_4_4/ai_54299259

From an MIR perspective, sampled audio is the most expensive method of storing information. Vast amounts of data exist for each piece, which leads to several difficulties when searching for query matches. For instance, if matching were done on the amplitude feature of a common musical piece recorded for 180 seconds at CD-quality, nearly 8 million data points will require investigation if the full, unreduced surface is used. Furthermore, searching accuracy may decrease with sampled music. Consider a query that is sung in a similar fashion to a recorded song; if background noise occurs in either of the recordings, mismatches may occur despite the correct performance of the user performing the query, due to the sensitivity of sound sampling at high frequencies. While sensitivity can be adjusted in theory, less research has been conducted that attempts to realize an MIR system which operates on sampled audio; instead, sampled audio is normally converted to a reduced form and searched thereafter. This trend is slowly reversing as of late, however.

2.1.2 Musical Instrument Digital Interface (MIDI)

Also widely available, MIDI is a numerical representation of music events, rather than a sampled recording like WAV or MP3. Specifically, MIDI tracks the beginning and ending of musical notes, divided into tracks where one track normally represents one instrument or voice within the musical performance. Each note in a MIDI note sequence is associated with a volume (velocity), a pitch, and a duration, and can also be modified by global track parameters such as tempo and pitch bending.

The key advantage of MIDI is that it relies on re-creating the musical performance, rather than actually recording how it sounds. Since the format is independent of samples, much less space is required to re-create the music. For example, consider a recording of a single note held for 2 seconds, say the note *middle C*. As a sampled recording, about 100,000 data points (samples) would be recorded; contrasting this with MIDI, which would require a *Noteon* event and a *Noteoff* totalling 8 data points or less, one can easily see that MIDI greatly reduces the amount of information required to reproduce music. While MIDI cannot exactly reproduce music as it sounded originally (due to a lack of timbre information), its intention is to provide a musical representation more similar to a digital score, rather than a sampled reproduction.

While most algorithms can search a song much faster in MIDI form rather than a sampled form, unprocessed MIDI itself is not commonly used in MIR systems. This is because MIDI also includes data that is normally discarded when searching amongst musical information, such as the tempo of a piece. Thus, many MIR systems convert MIDI/musical information to a representation that is MIDI-like; specifically, the notion of pitch and/or duration is copied, however extra information such as tempo or instrumental changes is often discarded.

2.1.3 Digital Score

Digital score approaches to musical representation attempt to closely mimic the oldest and most common musical form: the paper score. Digital scores try to encompass how music "should be played", instead of capturing how music was "actually played" as does MIDI and sampled formats. While there is a clear distinction in this regard, digital score formats actually function in a manner similar to a superset of MIDI; digital scores can record all of the musical expression available in standard MIDI, and also much more; most notably note division, time signatures, and note accidentals.

Theoretically, digital scores provide excellent candidate input for general MIR searching algorithms, as they successfully describe the fullness of musical expression in a very minimal way, thus being a very efficient format that can be easily and quickly searched. Despite this, very little music has been transcribed to digital score formats, with the main reason being that currently this process must be done by hand. MIDI would suffer from this problem also, except that it was designed to be more easily recorded through musical instruments that support the MIDI interface for fast and efficient input. Furthermore, while some sampled audio can be automatically transcribed into the MIDI format, there currently exists no flawless algorithm to capture sampled music into a *fully-featured* digital score; partial algorithms exist, but normally are error-prone. This is because MIDI considers much less information than a digital score, and information gathering from a sampled audio source has proven to be very difficult (see Section 3.0).

GUIDO description:

```
[ \clef<"f4"> \key<2> \meter<"4/4"> _*1/8
  \beam( d1 d d ) g0*1/4 \tieBegin b |
  \beam( b*1/8 \tieEnd e a g ) f#*1/4 d ]
```

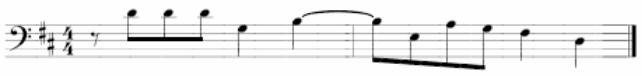

Figure 2
GUIDO Notation for a musical passage [Ren02]

One real-world example of a digital score format is the GUIDO Music Notation format[6] [HHRK98]. The format combines plain English keywords and tag-like structures to describe musical scores in a pseudo-graphical approach. The notation is normally read using a GUIDO parser, and although support has grown for the format in areas such as transcribed musical repositories and conversion tools, there are still far more freely available music files recorded in the MIDI file format. Still, MIDI can be converted to GUIDO notation and used with algorithms targeting GUIDO specifically. Despite this, very few researchers consider GUIDO due to a lack of industry standardization, and more importantly because vocally-specified user queries can prove difficult to translate to GUIDO; in general, transcribing voice to any format

[6] http://www.informatik.tu-darmstadt.de/AFS/GUIDO/docu/spec1.htm

from a comprehensive sampled format has its challenges, some of which are explored in Section 4.0.

2.2 Reduced Representations

Reduced representations attempt to target specific features of a musical composition or performance, in order to organize data in a manner that can be efficiently manipulated. One example of this would be the extraction of a pitch contour for a song; rhythmic information, tempo, instruments used, timbre, and all other information that does not describe the plot of pitch against time would be discarded, allowing an algorithm that relies solely on pitch data to perform directly on the representation.

While many ad-hoc representations exist that serve to highlight the strengths of particular algorithms put forth in MIR literature, most authors use one of the two reduced representations listed below. It is worth noting that when dealing with pitch and/or rhythm, many reduced representations use one of the representation alternatives outlined in Sections 2.3 and/or 2.4. Also, for an excellent tabular account of researchers that target the approaches in the subsections listed below, refer to the appendix in [CCI01].

2.2.1 String-based Representations

The string-based representation of musical features, normally pitch and/or rhythm, is by far the most common approach within MIR systems. Strings have a number of advantages:

- Reduces musical set to a very small amount of data
- Conceptual correlation between a single musical line and a single string
- Closely resembles the MIDI file format, which essentially is a set of numerically-based strings (when rhythm is discarded)
- Allows existing, mature string-searching knowledge to be adapted for use in music information retrieval; classic algorithms can be used such as the Knuth-Morris-Pratt (KMP) or Boyer-Moore (BM) algorithms [BM77, KMP77], as well as variations on these. This tends to lead to fast search times.

A wide range of authors make use of string-based representation(s) in their work, notably [CCIM+99, CILP+02, CIR98, CIMR+02, HCL01, NR98]. Typically, musical pieces (or excerpts of the original) are reduced to a string form and stored within a large database. Subsequent searches explore the string form rather than the original piece, which can normally be carried out online (at search time) within a short time frame.

Despite being the de facto standard approach for representing musical information in MIR systems, string-based representations do have limitations. First, while research into effective algorithms based on string-represented music is extensive, accuracy of results still remains a potential issue. This is normally a problem with the quantization process of either the original piece or the user query, due to the reduced alphabet size from which the string draws its symbols. Additionally, string-based representations are not particularly suited for polyphonic music. To represent multiple instruments operating simultaneously, several strings are needed, since one string can only track a single melodic line in a one-dimensional fashion. Thus, users that attempt to hum a musical theme are often in fact humming two different instruments operating in tandem, and matching algorithms that consider string-crossing greatly increase in time complexity. Should a string attempt to store several instruments together at one (as can be done using MIDI channels), this problem can be alleviated, though at the cost of introducing new problems such as string pollution (e.g. drums interrupting the melodic line). As a result of these limitations, recent research has targeted new approaches or modifications to string-based representations, to increase accuracy in the polyphonic domain.

2.2.2 Multidimensional Representations

A unique and effective approach identified by [MLW02] attempts to overcome the limitations of string-based representations by allowing for an *n*-dimensional space against which musical features are plotted, where *n* represents the number of distinct musical features that searching will attempt to resolve. Consider a searching mechanism that plots pitch and rhythm over time; these attributes create a 3-dimmentional space where all the notes of the piece would be plotted. Each set of plotted notes would then be stored in the database, one set for each song. The authors further develop this principle by designing an efficient algorithm for searching the n-space, while keeping their approach scalable to larger values of *n*, which is investigated in Section 5.4.6.

The main attractiveness of this approach lies in the inherent support for polyphonic music. Most feature-reduced representations, such as string-based ones, attempt to track one element of music such as pitch. While it is easier to match a query to a single feature, polyphonic music automatically adds an additional dimension upon which that feature relies. Consider a duet; pitch information is entirely dependent on which of the two parts that pitch is coming from, and in fact each registered pitch cannot be sensibly detached from this second dimension. Thus, polyphonic music is by nature a multidimensional entity, where the extraction of a single feature that is part-specific (like pitch, rhythm, etc..) is not possible without significant data loss.

The greatest challenge of multidimensional representations is not the storage of data, but more importantly the searching of that data. String-based

representations allow for very fast search response even in large databases, and multidimensional representations must attempt to match this speed, while at the same time strive for greater search accuracy.

2.3 Pitch Representation Alternatives

Pitch is undoubtedly the most-considered feature of music used by humans to identify a particular piece of music [Hof01]. Thus, it is worth outlining the various ways in which pitch can be represented, as well as a summary of the advantages of each alternative.

2.3.1 Absolute Pitch

Early crude MIR frameworks and systems adopted the idea of using absolute pitch to represent melodic lines of a piece. Storing the absolute pitch contour of a piece means taking the actual notes of a musical piece and assigning them a value based on their frequency. For example, consider the musical piece Frère Jacques:

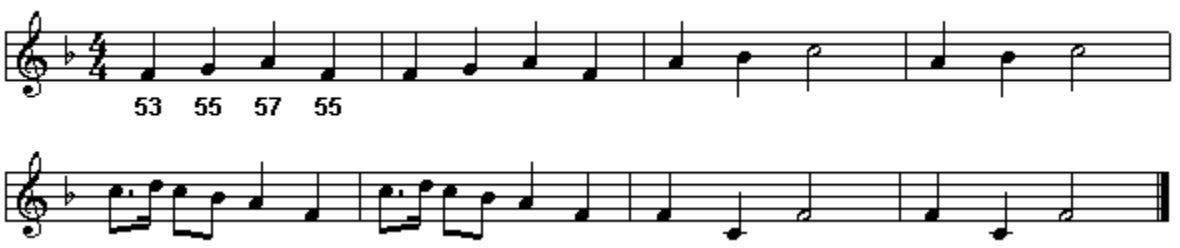

Figure 3
Score of Frère Jacques – First bar notated by absolute pitch using MIDI numbers

Most absolute pitch systems that label pitches with numerical values use the MIDI numbering system for musical pitch[7]. Thus, the first bar of this piece (Figure 3) contains an absolute pitch sequence of 53,55,57,55 in MIDI pitch format. These values can easily be stored as an absolute pitch string associated with a song in a database; thus, users can input a similar string of absolute pitches and attempt to find a match.

The problem with this approach is that it is insensitive to musical transposition. As an example, consider a user that sings the first bar accurately, but starts on an F note that is exactly an octave higher. Thus, the user's input query will be translated to the following pitch sequence: 65,67,69,65. While most would agree that the user is indeed singing the beginning of Frère Jacques, a mismatch would occur when 65,67,69,65 is compared to 53,55,57,55. Since very few users are capable of singing a query in exactly the key required (see Section 4.1), an MIR system using absolute numerical pitch values will often fail when searches are performed. This problem can

[7] http://www.harmony-central.com/MIDI/Doc/table2.html

be solved by checking that the distance between the query values and the target values are equidistant, however installing such checks would be computationally expensive in a largely-scaled database.

Another solution to this particular octave problem is to reduce the pitch alphabet to the 12 values representing the set of possible chromatic notes in common Western music {C,C#,D,D#,E,F,F#,G,G#,A,A#,B}; indeed, this is natural as most people already refer to music notes in their alphabetic form rather than numerical MIDI pitch notation. In this case, the first bar of Frère Jacques becomes F,G,A,F instead of 53,55,57,55, and the users' query also becomes F,G,A,F, leading to a match. Still, reducing the alphabet only works when transposition-matching is *exactly n* octaves apart; a transposition by any other value (such as a user starting a major third higher when singing) will not lead to a match. Thus, translating to letter values will only fix a small portion of all queries that do not have matching absolute pitches. For this reason, most MIR systems take an intervallic approach to pitch representation.

2.3.2 Pitch Intervals

There are many MIR systems that attempt to track the pitch feature of a musical piece through the use of pitch *intervals* rather than absolute pitches. Pitch intervals measure the distance between two subsequent pitches, often as a measure of semi-tonal steps between the two values. Positive values usually indicate a movement upward in pitch, whereas negative values indicate a movement downward.

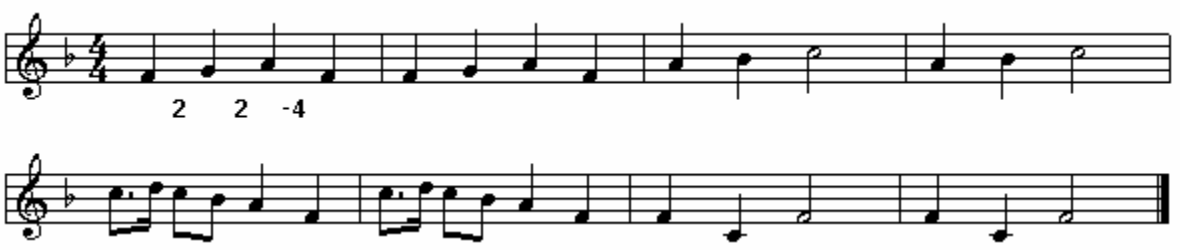

Figure 4
Score of Frère Jacques – First bar notated by pitch intervals

$$INTERVAL_Z = PITCH_{Z+1} - PITCH_Z$$

Equation 1

Consider Figure 4. The first bar of Frère Jacques has a pitch interval sequence of 2,2,-2 which reads "increase in 2 semitones, increase in 2 semitones, decrease by 2 semitones". These values can be obtained using Equation 1, where *z* represents an index position in a sequence of absolute pitch values and $PITCH_Z$ denotes the absolute pitch value at position z. This approach, unlike absolute value approaches, is much more robust in regard to musical transposition. For instance, if the Frère Jacques is

transposed upward by a major third, the first two bars of the piece will resemble those pictured in Figure 5.

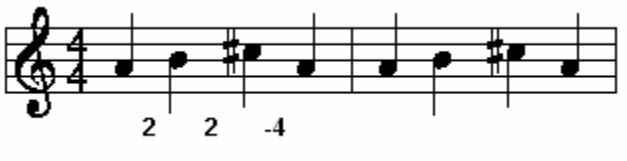

Figure 5
Score of Frère Jacques – Transposed up by a major third (4 semitones)

Thus, even if a user sings a rendition of Frère Jacques that is transpositionally equivalent to the stored version in an MIR database, a match will occur because the intervals between the notes will remain the same. While this illustrates the flexibility of using pitch intervals over absolute pitch, more advanced transpositions can still cause problems when using intervals, most notably scalar (or *diatonic*) transpositions from major to minor keys, which affect not only absolute pitch, but can change the intervals between pitches as well. Despite this, many researchers agree that changing a piece from major to minor (or, in general, changing the intervals between notes) is actually an alteration to the piece itself, or in essence changing the piece into a *new* piece, which would be perceived as somewhat different by human listeners. Thus, mismatches in this case are not only accepted, but expected. It is also interesting to note that pitch interval sequences can be manipulated (marginally) faster than absolute pitch sequences, since by definition they contain n-1 values where n = #pitches in the musical phrase.

2.3.3 Reduced Pitch Contour

A reduced contour is a quantization of pitch interval values over time to a small alphabet that represents the direction of the contour. This form of representation is very common in MIR literature, since it is easy to understand, fast to search (due to the reduced alphabet size), and shown to be somewhat effective (depending on the exact alphabet used) in differentiating melodic uniqueness. This can also be done in the rhythm domain, although it is not nearly as common an occurrence. While a potentially infinite number of reduced contours can exist, two well-known systems have been proposed, the latter of which has been used throughout MIR literature, and both are listed in subsections below. A variety of variations exist on the systems as well, to accomplish different types of matching; for example, see Section 5.3.2.

2.3.3.1 U,D,S Contour

One of the original papers that proposed using a reduced contour for MIR systems was authored by Ghias et al [GLCS95]. In this paper, pitch information was reduced to an alphabet size of 3:

- U – Indicates a rise in pitch during the note interval
- D – Indicates a drop in pitch during the note interval
- S – Indicates no change in pitch during the note interval

This approach is desirable from a workload perspective, as less effort is required to design complicated string matching algorithms that take into account larger alphabets. Furthermore, little work is required to convert a user query to the contour, since very simple transcription algorithms can be applied to determine the relative changes in recorded pitch with fuzzy granularity. The main drawback to this approach, however, is the lack of resolution that a 3-letter alphabet provides for the purpose of unique identification of musical passages. This is further explained in the conclusion of the authors' paper, which suggests that a larger alphabet is desirable.

2.3.3.2 Step-Leap Contour

The step-leap contour is the most used reduced contour in MIR literature, requiring an alphabet of 5 distinct values to describe a pitch interval contour. It strikes a balance of increased resolution and loss of efficiency, when compared to the original U,D,S approach by Ghias et al. A typical step-leap alphabet consists of the following values:

- -S – Indicates a small drop in pitch during the note interval
- -L – Indicates a large drop in pitch during the note interval
- R – Indicates no change in pitch during the note interval
- S – Indicates a small rise in pitch during the note interval
- L – Indicates a large rise in pitch during the note interval

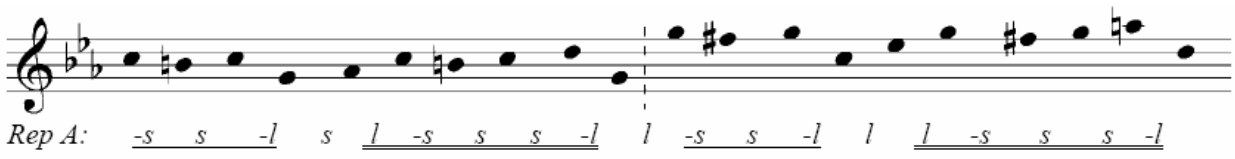

Figure 6
A step-leap contour representation of a musical passage [CT04]

Cambouropoulos and Tsougras make use of this representation for their melodic segmentation algorithm PAT [CT04]. Figure 6, adapted from their paper, shows a musical passage annotated by step-leap values. In this figure, a step is defined as a change of pitch with a magnitude of either 1 or 2 semitones, whereas a leap is any change in pitch great than this. No notes are repeated in this segment, and so no letter representing zero pitch change is used (no R's appear). The increase in resolution from 3 to 5 letters allows for greater matching potential for the identification and differentiation between varying musical passages, and yet string matching algorithms can remain simple and efficient overall. The step-leap contour

approach can also be further extended, to allow for multiple classes of steps and leaps, with overlapping categories as well [CT04].

2.4 Rhythm Representation Alternatives

Like pitch, rhythm is a common musical feature used for data retrieval within most MIR systems, though to a lesser extent. There are three common approaches used by MIR systems and researchers, each of which are outlined in the sections below.

2.4.1 Absolute Duration

Similar to absolute pitch (2.3.1), absolute duration measures the exact lengths of notes that occur within a piece.

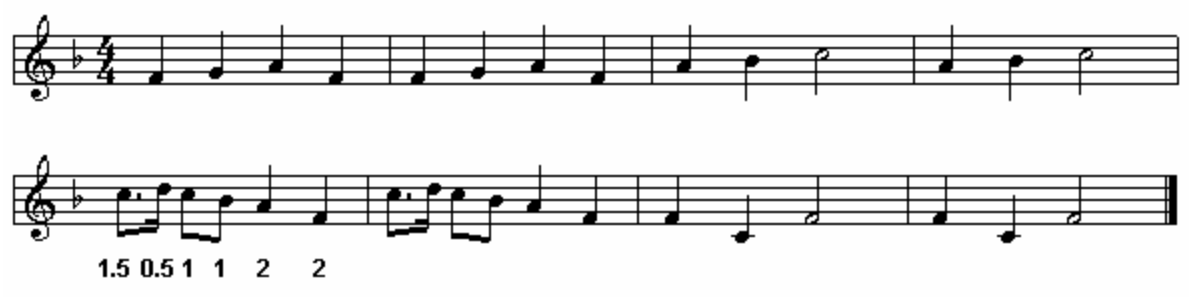

Figure 7
Score of Frère Jacques – Fifth bar notated by absolute duration (eighth note = 1)

The numerical sequence representing the rhythmic feature of the fifth bar of Figure 7 assumes that eighth notes have a numerical weight of 1; that is, durations which take twice as long (quarter notes) have double the weight, whereas durations which take half the time (sixteenth notes) have half the duration. In practice, this scheme usually works well. Unlike absolute pitch, absolute rhythmic sequences do not suffer from the pitch transposition problem; as long as a user sings at a steady pace, rhythm can be successfully matched regardless of the piece's tessitura. Despite being a more useful absolute measure than its pitch counterpart, absolute rhythm does run into problems when a piece is time-stretched.

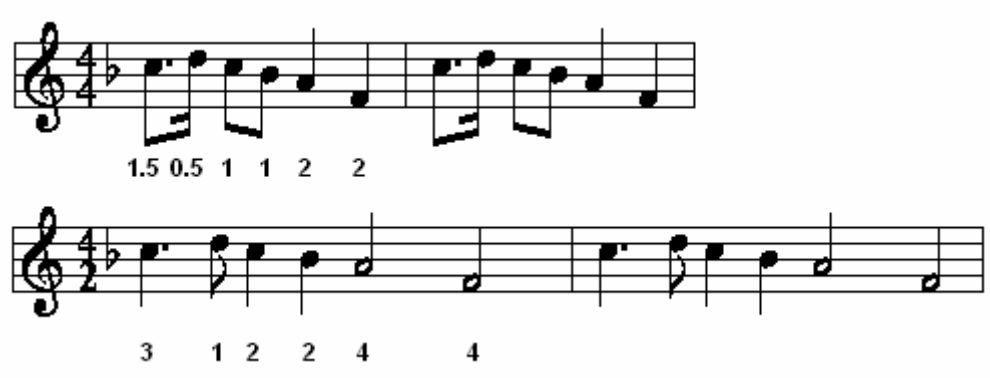

Figure 8
Two scores of Frère Jacques – Fifth and sixth bars (fifth with absolute duration values)

Figure 8 shows the fifth and sixth bars of Frère Jacques. The first passage uses exactly half the weighted duration that the second passage contains; that is, the second passage is time-stretched by doubling the values of each individual note. Audio playback of either of these passages will sound familiar to humans, and both would be correctly identified as a part of Frère Jacques. Despite this, a mismatch may occur if a user sings this passage as a query, even if sung tonally correct. This is because it is difficult to ascertain note division from recorded audio input alone. In fact, when played using the same tempo (say, from a metronome), there would be no audible difference between these two passages. Thus, when a musical query is input there may be several ways to encode the rhythmic sequence, all of which would be correct, and will ultimately lead to mismatching. It is quite possible that the second passage shown was a user-given query attempting to match the first passage, and since 1.5,0.5,1,1,2,2 will does not match 3,1,2,2,4,4, an MIR system using absolute rhythm would fail. To overcome this problem without resorting to mathematical re-calculations on entire musical segments, duration ratios are often a better choice to represent rhythmic features.

2.4.2 Duration Ratios

Duration ratios attempt to rectify the problem of time stretching suffered by the use of absolute duration values. Though there is no standard way of calculating these ratios, Equation 2 shows one possible way to achieve duration ratios, where *z* is an index position in a sequence of absolute duration values and $NUMBEATS_Z$ denotes the number of beats that the absolute duration of the *z*-th note occupies within a bar:

$$DURRATIO_Z = \frac{NUMBEATS_{Z+1}}{NUMBEATS_Z}$$

Equation 2

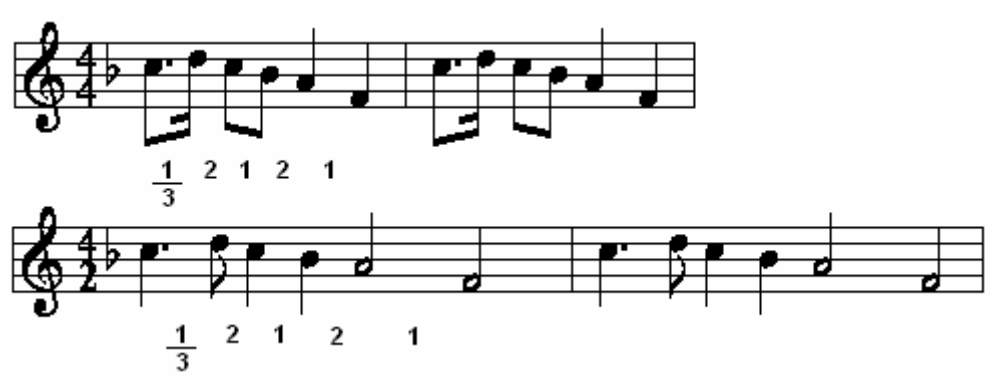

Figure 9
Two separate scores of Frère Jacques – Fifth and sixth bars (fifth with duration ratios)

Figure 9 shows that the effects of stretching time have no bearing on the ability for matches to occur; beat time can be expanded or compacted in either the numerator or denominator of the piece's time signature, with no effect to the resulting duration ratios. Although duration ratios are more useful than absolute duration values for the purpose of an MIR system, both of these schemes still suffer from real-world problems; in particular, the problem of ambiguous duration transcription from recorded music. Consider an MIR system where querying is done by use of captured MIDI from a piano/keyboard. Say that we have stored the song "Hot Cross Buns" (along with duration ratios according to Equation 2) in our database as seen in Figure 10.

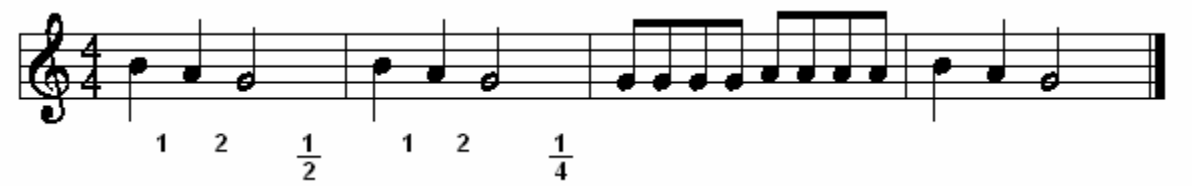

Figure 10
Score of Hot Cross Buns – First and second bars annotated with duration ratios

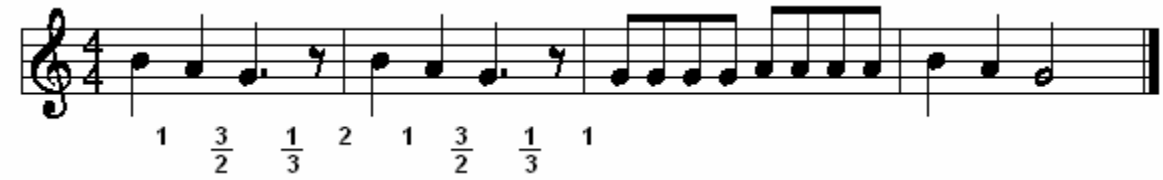

Figure 11
Score of user query – First and second bars annotated with duration ratios

Furthermore, let Figure 11 represent a user's query (played on a keyboard) who is attempting to find the score for Hot Cross Buns. In this example, the user has played the piece in a seemingly correct fashion; the only minor error made was that their half-notes (notes 3 and 6) were not held for the full required duration called for

by the original score of Hot Cross Buns, a common mistake made by all musical performers. Still, since MIDI is sensitive to such anomalies, the users' query will be recorded exactly as played; it would make little sense to apply a heuristic here, since making assumptions about note durations is nonsensical given that music is infinitely divisible, and sometimes divides into fine granularity (64^{th} notes) in practice.

Very quickly we see that a mismatch is going to occur, since the duration ratio sequences of the two passages are clearly different. A partial match will occur if the third and fourth bars are considered, however for a full record match to be declared, the MIR system will have to allow tolerance for individual mismatches; more accurately, two mismatches for every note duration misplayed during the query. As a result, many MIR systems opt for an alternative approach that uses *Interonset* Ratios, rather than duration ratios.

2.4.3 Interonset Ratios

Interonset ratios (IORs) are an excellent way to represent rhythmic information while bypassing the drawbacks listed in Sections 2.4.1 and 2.4.2. Most recent research utilizes interonset ratios when tracking rhythmic features within an MIR system. It is first useful to define an interonset interval, and to show the advantage of using these to create interonset ratios.

$$IOI_Z = ONSETTIME_{Z+1} - ONSETTIME_Z$$

Equation 3

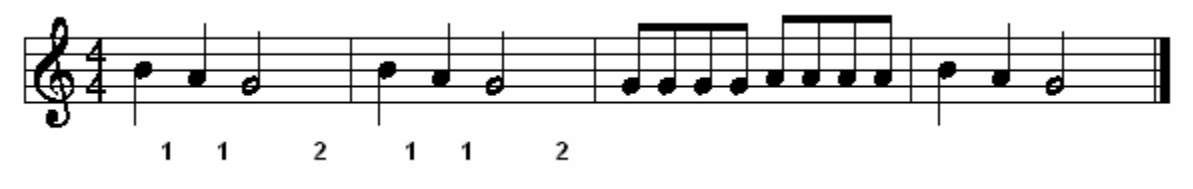

Figure 12
Score of Hot Cross Buns – First and second bars annotated with IOIs

An IOI totally ignores a note's length, and instead tracks the time between the beginning points of two successive notes. Equation 3 shows the construction of IOIs, where the $ONSETTIME_Z$ of a note is a measure of how many beats have passed since some point of reference (such as the beginning of a piece or bar). The sequence of IOIs in Figure 12 is 1,1,2,1,1,2 for the first two bars of the piece; this corresponds to a difference of "1 beat between the starting times of the first two notes, 1 beat between notes 2 and 3, 2 beats between the notes 3 and 4, etc..".

Tracking rhythm in this manner leads to a both positive and negative effects within an MIR system. On the positive side, the user query problems discussed in Section 2.4.2 (in reference to Figure 11) no longer cause mismatches within the system; if a note is held a little longer or shorter than the same corresponding note in the database, IOIs will not be sensitive to it. However, this can also be seen as a negative side effect; if a user were to search for a song containing specific note durations, say staccato notes, IOIs would not be appropriate. While this could present a problem for some systems, most authors agree that trading off duration sensitivity in an effort to relax query requirements of rhythmic perfection (see also Section 4.4) is acceptable. A second problem exists with using IOIs, and also exists with the use of absolute durations as well, where the values are not sensitive to time stretches. For this reason, calculated IOIs can be converted to interonset ratios, or IORs, which are sensitive to both time stretching and also imperfect user query note durations.

$$IOR_Z = \frac{IOI_{Z+1}}{IOI_Z}$$

Equation 4

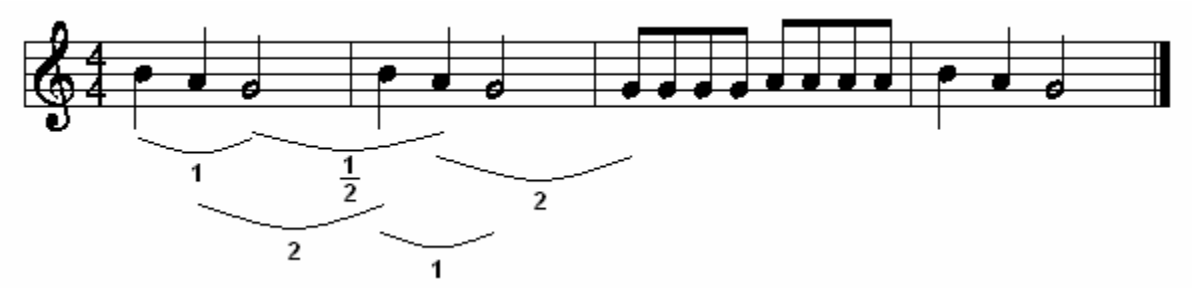

Figure 13
Score of Hot Cross Buns – First and second bars annotated with IORs

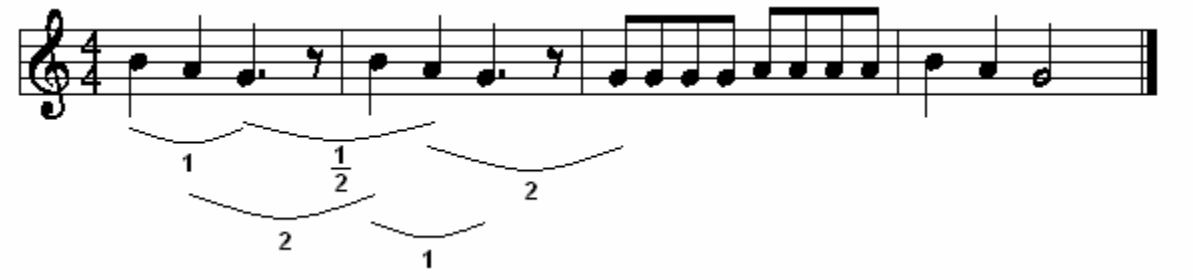

Figure 14
Score of user query – First and second bars annotated with IORs

Using the formula denoted in Equation 4, Figure 13 shows the rhythmic component of Hot Cross Buns stored as a sequence of IORs 1,2,0.5,1,2. This can be

read as “the first interonset interval and the second interonset interval are different by a factor of 1, the second and third by a factor of 2, etc..”. Lartillot notes that IORs are far less intuitive a measure than the representations described in previous sections [Lar05]. Despite this, they still accomplish the goals of being sensitive to duration variance and time stretching. For example, the IOR sequence of a sample user query in Figure 14 will find a match even if the durations vary slightly and/or time is stretched (in this example time is not stretched, however).

2.4.4 Reduced Rhythmic Contour

A reduced rhythmic contour is rarely used in the academic MIR literature. This is mainly due to the general lack of concentration on rhythmic features when compared to pitch features, for the purpose of pattern induction and searching. Still, reducing the rhythmic quantization alphabet to a very small size can have advantages over larger alphabets with more precision, and this approach could be particularly suited for bit parallelism (see Section 5.4.3), though this literature survey found no papers with such a solution. In an effort to avoid repeating the key points of reduced contours, please refer to the discussion in Section 2.3.3.

3.0 Musical Transcription

This paper defines musical transcription as the process of converting music from one form into another, specifically limited to the domain of computer-based representations. Musical data can be expressed in a variety of formats, many of which have gained widespread usage on the internet. Thus, MIR systems often have to deal with the problem of transcription to achieve a common form for music to be represented. The main difficulty in any transcription process is the interpretation of music from the source form, before converting to the target form. The two most common forms of transcription relate to the conversion of music from a MIDI source format or a sampled audio format, both of which are described in Sections 2.1.1 and 2.1.2.

Music can be challenging to transcribe for a variety of reasons inherent to musical performance itself. First, a great deal of musical knowledge is needed to properly recognize all of the features that can (or need to) be extracted. Most computer scientists or mathematicians designing transcription algorithms must therefore consult experts in musical theory, further complicating the process [MB01], unless they are fortunate enough to possess this musical knowledge themselves. Additionally, music contains a large amount of invisible data that is passively implied rather than explicitly stated in source form. For example, subtle key changes can be implied through the use of note accidentals, and a piece's chordal progression is merely implied by the relationship of notes to a tonic value. Finally, musical transcription is greatly complicated in the case of polyphonic interplay, where distinguishing between several instruments playing concurrently can prove to be very challenging.

3.1 Transcription from Sampled Audio

Transcription of music from a sampled audio format can be very complex depending on what musical information needs to be extracted and converted. As a result, this paper merely brushes upon tactics used for transcription from sampled music; entire surveys exist on these topics alone, and many of the details are omitted in this section as a result. Early attempts at transcription (~1970) commonly targeted the speech domain, rather than music, as there was great interest in being able to speak into a microphone and have a computer "type for you", or vice-versa where the roles of speaker and writer were reversed. Hess's algorithm for speech transcription was one of the first attempts to apply a transcription algorithm to the musical domain, with very limited success[GKM03]. Before that, Piszczalski and Galler developed one of the first systems that attempted to convert an audio signal into musical score [PG77]. Later in the early and mid 1990's, a growing set of approaches specifically targeted for music began to emerge.

Most algorithms focus on the extraction of time-domain specific fundamental frequencies found within the musical waveform. A fundamental frequency, or simply *fundamental*, is defined as the lowest frequency value in a related harmonic sequence of frequencies, where each value in the series is a multiple of the fundamental. By extracting this value, analysis can be performed to provide a pitch spelling, which is an ordered (time-specific) listing of the various pitches that occur throughout the piece. Two fairly common approaches attempt to identify a periodic signal's fundamental frequency, and lend to additional musical information:

- **Autocorrelation (ACR)** – This method is the most common approach at the identification of periodic fundamental frequency [Kla00]. Given a sequence of frequencies sampled from an audio signal x(k) of length K, an autocorrelation function ACR(n) is defined as:

$$ACR(n) = \frac{1}{K} \sum_{K=0}^{K-n-1} x(k) \cdot x(k+n)$$

Equation 5

  This function will produce an ACR contour, where the peak value of the result represents the fundamental frequency of the original sequence. Variations on this function also exist, and these tend to increase or decrease the sensitivity of results, making autocorrelation an effective tool for pitch spelling.

  Another property of an ACR contour is that frequency punctuations can be followed over time, which is useful for beat tracking. Still, while beat divisions can be somewhat implied, further analysis is needed to determine exactly where bar lines can be drawn. For example, if a large number of polyphonic frequencies fall within a close period of time, and they do so at regular intervals (revealed by the ACR contour), then a possible bar division may be extracted.
- **Envelope Periodicity (EP)** – This method capitalizes on the observation that musical signals with multiple frequencies occurring simultaneously in the time-domain often cause fluctuations in the corresponding amplitude envelope, where the frequency of each fluctuation is a measure of the difference between the observed frequencies occurring simultaneously [GKM03]. Since the fundamental frequency will tend to affect the amplitude envelope the strongest, it can be identified, as well as potential information about the rhythmic beat of the music.

Aside from these and many other approaches, there are additional considerations when transcribing musical data that go beyond pitch and rhythm factors. For example, the identification of time and key signatures can be challenging even given a perfect transcription of pitch and rhythm, as often musical passages can

be broken down in a variety of equally-correct ways. Some of these additional considerations are further explained in the following section (3.2).

3.2 Transcription from MIDI

When compared to transcription from sampled audio, MIDI tends to be a much nicer format to work with. Essentially, MIDI resembles a form that most of the algorithms in Section 3.1 attempt to achieve; a quantized mathematical decision on each and every note in a musical phrase. Still, MIDI can be difficult to process further in cases where higher-level musical information (such as key, bar segmentation, etc) is required. Some of this difficult depends on the quality of the MIDI file. For example, were separate channels used for the various instruments involved, or were all of the notes crammed into one channel? Does the file follow general MIDI standards, such as the use of channel 10 for drums? If a MIDI file is not properly designed, the task of transcribing the music to a more detailed format (such as a digital score) becomes harder. The following sections outline some of the challenges of transcribing MIDI music into alternate, more detailed forms.

3.2.1 Pitch Spelling

Generally, pitch spelling from MIDI is a simple task, since all notes in a MIDI file have clearly-defined pitch values ranging from 0-128 with chromatic semi-tonal granularity. There are some considerations that can complicate the process, however. First, consider a 3-note phrase of increasing MIDI pitch numbers corresponding to 69,70,71. Using a simple MIDI pitch lookup table[8], one would translate these values into the notes A, $A^{\#}$, B (Figure 15).

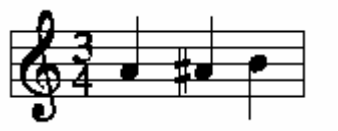

Figure 15
A pitch spelling of MIDI values 69,70,71

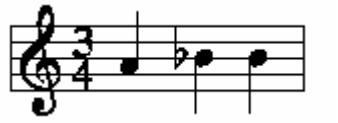

Figure 16
An alternate pitch spelling of MIDI values 69,70,71

[8] http://www.harmony-central.com/MIDI/Doc/table2.html

However, Figure 16 shows an alternate pitch spelling which is equally valid, using a flat-augmented B instead of a sharp-augmented A, both of which are enharmonic notes. In fact, there are other potentially correct spellings of 69,70,71, that use notationally parsimonistic values, such as double-sharps or double-flats. The problem lies in the interpretation of these intervals as "chromatic" intervals as opposed to "diatonic" intervals. Chromatic intervals measure the number of semitones between two values, and in the case of Figure 15 and Figure 16, the chromatic intervals are identical. Diatonic intervals measure distances between note values that fit nicely into common major or minor scales, and so certain intervals carry heavier weight than others when pitch spelling occurs. Most authors design their algorithms to avoid double-accidentals, as they occur infrequently in real-world musical scores.

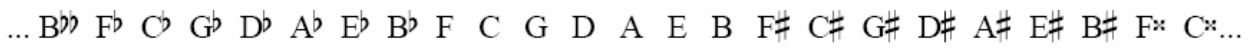

Figure 17
The "line of fifths", adapted from [Cam01].

Two similar algorithms presented by Temperley and Longuet-Higgins attempt to exploit a linear representation of the "circle of fifths"[9], termed the "line of fifths"[10] (see Figure 17) [Lon87, Tem97]. In both algorithms, the spelling of two subsequent pitches minimizes the interval distance between the note values as they occur on the line of fifths. This gives a natural pitch spelling where oddly-occurring intervals are avoided, and while both algorithms apply further analysis, they achieve similar (and effective) end results. Additional authors have since taken the approaches defined by Temperley and Longuet-Higgins and attempted to improve them, such as the *ps13* algorithm by Meredith [Mer04].

Class	A	B				C				D		
Intervals	4P	2m	2M	3m	3M	2a	3d	4d	4a	1d	3a	2d
	5P	7M	7m	6M	6m	7d	6a	5a	5d	1a	6d	7a

Figure 18
Cambouropoulos' Pitch Classes
(P=perfect; M=major; m=minor; a=augmented; d=diminished)

Other pitch spelling algorithms exist that attempt to achieve a proper pitch spelling where chromatic ambiguity occurs. For instance, Cambouropoulos suggests the use of diatonic intervals to achieve accurate pitch spelling [Cam01]. His approach is based upon the classification of standard and non-standard diatonic intervals that can occur in music, and he identifies four pitch classes A-D, where each interval is assigned a class, and higher alphabetically-ordered classes carry greater precedence (Figure 18). The algorithm then spells out all possible note spellings for each bar (or window) of music between 9 and 15 notes, and assigns pitch classes to the resulting

[9] http://en.wikipedia.org/wiki/Circle_of_fifths
[10] http://www.music.sc.edu/fs/bain/atmi02/lof/index.html

possibilities. The spelling that maximizes the alphabetic ordering (generates the highest classes) is then selected as the proper spelling for that particular window of music. While a great deal of processing is required (up to 5000 spellings per bar of music), accuracy is quite high and leads to spellings that conform to many of the best human practices of musical score construction.

3.2.2 Polyphonic Division

Like pitch spelling, polyphonic division is normally a simpler process with MIDI data than with sampled audio. The standard MIDI file format accounts for the division of music into channels; conceptually, a channel (usually) corresponds to one instrument (sometimes able to play several simultaneous notes, such as a piano), making a channel synonymous with a musical "part". Thus, polyphonic division is natural; the final score adds a staff for each channel presented in the MIDI file, and transcription begins with algorithms such as those identified in Section 3.2.1, many of which are capable of handling polyphonic transcription themselves. However, some MIDI files are not structured in this manner, and instead opt to merge several parts into one channel. These malformed files still contain all of the information to recreate the sound of a musical piece, yet they are difficult to parse for the purpose of accurate polyphonic transcription. Consider a 2D orthogonal projection of a MIDI channel, where (x, y) denotes the occurrence of pitch x at time y within the piece.

$$\{\ (50,1),\ (52,1),\ (52,2),\ (53,2),\ (53,3),\ (55,\ 3)\ \}$$

Figure 19
A set of 6 MIDI notes found in a single MIDI channel in (pitch, time) format

The set defined in Figure 19 outlines 6 differently-ordered notes that could occur in a MIDI file with only 1 channel. It is difficult to determine how to interpret these values. All six notes may be part of a single piano part (Figure 20), or the notes could be divided into two different piano parts operating together in multi-voiced polyphony (Figure 21).

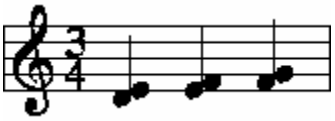

Figure 20
Single-voiced pitch spelling based on note values of Figure 19

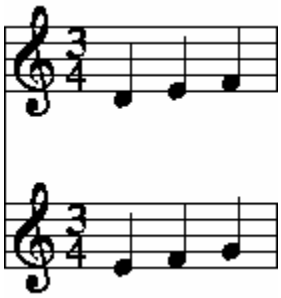

Figure 21
Multi-voiced pitch spelling based on note values of Figure 19

In cases such as these, it is difficult to know when notes should be separated and when they should not; for the purpose of pattern matching (see Section 6.0), it is critical that notes not be separated outside of their polyphonic context [Meu03]. The simplest solution is to assume that MIDI files are correctly formed, and thus side with the monodic translation. While this is probably the most correct approach, it leads to transcription errors for some MIDI files, and the application of heuristics in this area is still relatively unused, with the exception that additional piece information (such as genre, composer, etc.) is available.

An attempt at polyphonic division by Cambouropoulos applies a streaming algorithm to polyphonic music [Cam00b], so that each vocal part within the polyphonic phrase is broken down into a single steam, or voice, normally for the purpose of creating a set of note strings to which matching algorithms can be applied. The algorithm used breaks down musical passages into *n* streams of notes, where *n* is the number of notes within the largest chord that occurs during the piece. Despite some success, this approach is naïve from a musical standpoint; no crossing of streams is allowed, which is a phenomenon that occurs frequently in real-world music. Thus, mismatches can occur as illustrated in Figure 22, though enhancements made to the algorithm can improve accuracy.

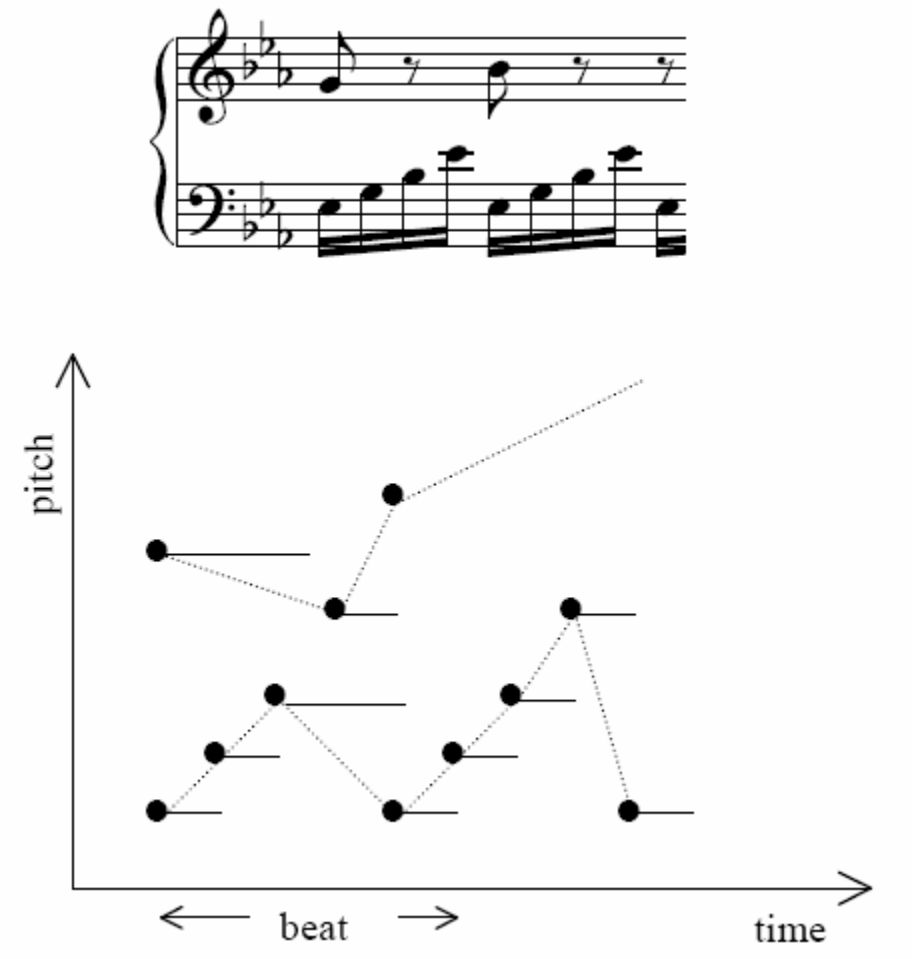

Figure 22
(Top) Two musical voices as seen by a musician playing music
(Bottom) Incorrect streaming of voices by Cambouropoulos' algorithm [Cam00a]

3.2.3 Beat Induction

Human beings have an inherent understanding of the natural division within a musical piece that tends to divide notes into even periods of time, or bars [Cam00b]. To translate this innate ability to a computer system, many MIR systems attempt to analyze music into sections or windows, where the beat is identified as the "best-fit" time division of some or all of the notes contained within the window. This can be determined by collecting all of the interonset intervals (IOIs) between subsequent notes, and then grouping the intervals into common categories, akin to an IOI histogram [DC00, MM04]. Once all of the intervals have been grouped, the most common category (containing the most IOIs) is identified as the beat interval, and thus induces the overall beat of the musical piece. While this approach is regularly used, other variations and adaptations exist such as Large and Kolen's model utilizing interonset values to form nonlinear beat oscillators [LK94].

In this process, called *beat induction*, it is important to determine how musical beat is identified in relationship to the notes participating in the induction algorithm. Two approaches are often used, each of which tends to achieve reasonably successful results when applied to most forms of music.

- **Simple Note Division** – In this approach, any and all notes within the time window are considered equally important in identifying the presence of musical beat [Ros92, Row93]. Thus, each IOI carries equal weight, and as mentioned above, contributes to the weight of an IOI category in an equal manner. This tends to work reasonably well for sufficiently large musical windows, as note onsets tend to define beat over time even when strange rhythmic values are encountered. For example, consider a fermata note augmentation; the corresponding IOIs associated with the notes before and after the fermata will likely outline an odd and lengthier interonset interval, thus falling into an uncommon category that will not dramatically affect the beat induction process.
- **Accentuated Note Division** – This model attempts to distinguish between certain notes that should contribute with greater weight to the induction of musical beat. Many authors identify notes based on their accentuation value, or *salience* [Kla03, LJ83, Par94, PE85, Sch98, Sep01].

Figure 23
Visual description of note accentuation according to Povel and Essens[11]

Figure 23 shows an example accentuated and non-accentuated notes in a musical window. In this particular example, the following rules define which notes should be accented:

- Isolated Tones
- Latter tone in a two-tone group
- First and last tone in a 3+ tone group

By associating higher weights to accented notes, it is possible to avoid inappropriate induction on runs of notes that operate outside of the regular beat. For example, consider the case of musical arpeggios (Figure 24). Using the rules above, the disruption of an intended arpeggiated chord played as a series of four distinct

[11] http://www-classes.usc.edu/engr/ise/599muscog/2003/week12/Chen-TempPatterns.ppt

notes is reduced, since heavier rhythmic weights are tied to the first and last notes in the run, rather than treating all of the notes equally when inducing beat.

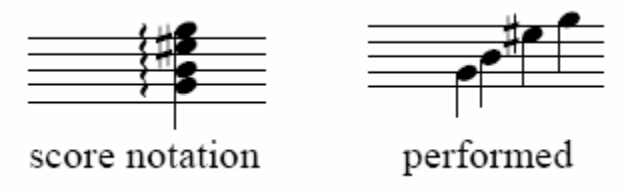

Figure 24
An arpeggiated chord and its corresponding real-world MIDI notes (adapted from [Cam00b])

3.2.4 Key Signature Identification and Harmonic Analysis

$$P(k \mid m) = \frac{P(m \mid k)P(k)}{P(m)}$$

Equation 6

Two potentially important areas of musical transcription have been relatively ignored within MIR literature, probably due to the constant focus on transcribing pitch and rhythm based on their prominence in MIR systems. The induction of key signatures from music has the potential to assist pitch spelling algorithms in the correct identification of diatonic note values, and when combined with harmonic (chordal) information, can form an excellent basis for a transposition-invariant algorithmic music generation algorithm. Klapuri outlines a simple model for the determination of key signature at any given point in music based on a single note value, using the standard Bayesian formula shown in Equation 6, where *m* represents the note element and *k* represents the key element [Kla03]. Based on findings by other authors, Klapuri also models how a key can be probabilistically determined by several notes occurring in polyphony.

In the same paper, Klapuri also outlines a model for determining the harmonic progression of a musical passage, based on a nearly identical Bayesian model as that used to identify key signature. Additional input to this model can be a key signature itself, adding to the probable weight of each possible chord to be considered during the chordal spelling. While the paper provides a fresh look at the problem of transcribing key signature and harmonic progression information from musical passages, there is plenty of add additional room for research within the academic community on these topics.

4.0 Query Capture and Representation

A key component of any MIR system is its ability to convert a user's search intentions into a usable form for the purpose of record matching. This is a very difficult task, since music itself is complex and difficult for many (if not most) users to express. Furthermore, it is nearly impossible to identify a single query method that works for all types of musical searching, as any one mechanism would likely fail to consider the wide range of musical features inherent in each piece. Thus the selection of a query mechanism can greatly affect what an MIR system can actually accomplish, and there exists a balance to be struck between the undesired complexity and desired flexibility of using multiple methods for query capture.

An even greater challenge for MIR systems is their ability to compensate for lack of user skill [WR03]. While text-based systems are sometimes intolerant of mistakes (spelling, grammar), they are accepted as so because users are expected to be able to spell correctly, or at least very close. Conversely, it is not reasonable to expect a user to express music in a correct fashion, as mistakes are far more abundant in the music domain due to the complex nature of the medium. Furthermore, the automatic correction of these mistakes is an important yet challenging task, as a pre-defined dictionary of possible input does not exist in the music domain as does in the textual domain. Some of the common challenges that must be met by query input components of MIR systems are:

- **Vocal Pitch Drift** – In any query system that requires the use of human vocal chords, pitch drift can often become a problem, particularly as a query grows in length. Even skilled musicians often have difficulty expressing musical queries in a flawless fashion for sustained periods of time [UZ98]. Thus, while a user may start a query on key, that key may slowly change (normally upward), to the point where a particular note at the beginning is sung at a much different frequency than the same note at the end of the query. This often encourages MIR systems to limit query length, in an effort to reduce the effect.
- **Personal Tonic** – Some musical instruments, while tuned correctly in regard to themselves, are not tuned according to the standard frequencies stated for common Western music 12-tone scales [Ada04]. Furthermore, it is rare that a singer begins a query with perfect pitch, where their own tonic matches a standard note frequency[12]. Consider a query of three notes, with frequencies 430Hz, 513.25Hz, 577.33Hz. While the intervals between these notes match the intervals between the musical sequence A_4-C_5-D_5, the frequencies do not match the correct values stated for these notes. Thus, the input device has its own personal tonic, and the problem essentially becomes a quantization issue. While a simple pitch shift of the entire query could be conducted, the intervals between the notes themselves could occur in such a manner that ambiguity

[12] http://www.phy.mtu.edu/~suits/notefreqs.html

exists as to how and when the pitch shift could occur, further complicating the process.

- **User Skill (Incorrect Values)** – This problem applies to any form of query input, from simple tapping to complex notational input; problems such as pitch drift are also a factor of user skill, but differ in that the query is essentially correct, but modified in a uniform way that can be potentially detected and corrected. Because it is difficult to correct wrongly-stated notes in queries, many MIR systems opt to design tolerant matching algorithms, some of which are covered in Section 6.0. One such approach explored by Dowling found that users are often more apt to remember the melodic contour of a piece, rather than actual note intervals [Dow78]. Thus, an MIR system using a Step-Leap representation (Section 2.3.3) may be more tolerant to poor user skill and still be useful for matching purposes.
- **Pitch Repetition** – This problem affects the domain of human vocal input. Users often prefer to repeat a pitch for sequences of notes that actually should differ. This normally occurs when a user does not fully know how the melody goes (some fuzziness in memory), and so they hum or sing the same value until a part is reached where memory kicks in and greater intervals begin to occur. In this case, it can be difficult for an MIR system to determine the point at which a user is "lost", given that they may be intentionally repeating values.
- **Monophonic Input** – This problem can be illustrated by imagining a user attempting to find his or her favourite choral piece. The user specifies a query, but can only sing one part (a limitation of the human voice); in choral music, there may be many vocal parts stored as a polyphonic aggregation within the MIR database. Thus, the user's query is limited because they can only produce one note at a time. This problem can affect vocally-based queries, as well as many instruments (such as the flute) that can only play one note simultaneously. Rather than allowing the user to input several lines as part of a multi-step query that can target polyphony, most systems wisely attempt to match the single query line to the most significant portion of the polyphonic target to obtain a match.
- **Tempo Creep/Stutter** – Users specifying queries with a rhythmic component often speed up, sometimes dramatically, while they produce music. This is often a result of poor internal beat, where the use of a visual metronome can sometimes provide assistance during the query process. Additionally, tempo stuttering sometimes occurs when a user makes an error or forgets the next notes in a musical query; the end result is a temporary loss of sustained tempo, and MIR systems can meet this problem using relaxed rhythmic matching or correlation techniques.
- **Articulation Issues** – Notes within a vocal query (possible for some instrumental queries as well) can be inappropriately merged or separated by a query capture mechanism due to issues in articulation. Sometimes a user may produce notes with an "oooo-ing" noise, making the distinction between when a note ends another begins difficult to detect; this is especially evident when two subsequent notes have the same pitch. Furthermore, the opposite effect can occur when users sing a query that contains lyrics; the two syllables of the

word “never” may cause the query capture mechanism to think that the user is specifying two separate notes, when in fact words like “never” are often associated with a single note in popular music. Adams suggests that if the user articulates passages with a nonsense word, such as “da” or “ta”, a query capture system can better determine note division [Ada04].

- **Artistic Impression** – Users may be compelled to add their own flair to their queries, in the form of note embellishments, tempo throttling, or even (though less likely) newly-improvised passages. This can add considerable difficulty to any query processing mechanism [WR03], and may result in unusable input.
- **User Perception of Music** – In work by Krumhansl and Shepard, it was found that the musical experience or skill level of a user can affect their actual perception of music, such as whether or not they are capable of discerning note intervals [KS79]. This and other perception issues are further explored by Uitdenbogerd and Zobel, who consider query difficulty from a human psychoacoustic perspective, and identify problems that can begin before a query is even stated, such as a user’s inability to determine how many parts are operating simultaneously in fast-moving musical phrases [UZ98]. The net effect of poor musical perception is that users will not be able to reproduce music correctly, given that their psychoacoustic model of the musical passage is broken.

The sections listed below outline some of the aspects and challenges associated with various forms for querying. For information on how queries can be used and compared to entries within an MIR database, see Section 5.3.

4.1 Query by Humming

The collection of various input methods requiring use of the human voice is typically referred to as “query by humming”. A very large portion of MIR research has focused on this paradigm; an interesting selection, considering that hummed queries can suffer from the articulation problems identified above, and also that vocal transcription has proven difficult in the music domain [Par05]. Use of human voice has several advantages over other query paradigms, despite the previously-listed drawbacks. First, the human voice is the only instrument usable by any and all human beings at any time, reducing the requirement that an MIR system be accompanied with some separate input instrument. Second, the human voice is the most widely used instrument, maximizing the potential number of users due to the universal experience that humans have in wielding their own voice. Finally, by developing robust algorithms to handle vocal signals, a system can be easily adapted to use other forms of instrumental input, since the human voice presents many challenges that form a general superset of the drawbacks of most other instruments, at least in the monophonic domain.

Query by humming was popularized by Ghias et al. in 1995 when they proposed a novel MIR system using a simple hummed melody as input into an musical database,

where the query signal would be converted into a sequence of directional characters outlining a pitch contour [GLCS95]; the same approach defined in Section 2.3.3 using the alphabet {U,D,S} where U means higher or "upward" pitch, D means lower or "downward" pitch, and S represents an interval with zero pitch difference. Figure 25 shows the process as envisioned by the authors, from microphone to end-result.

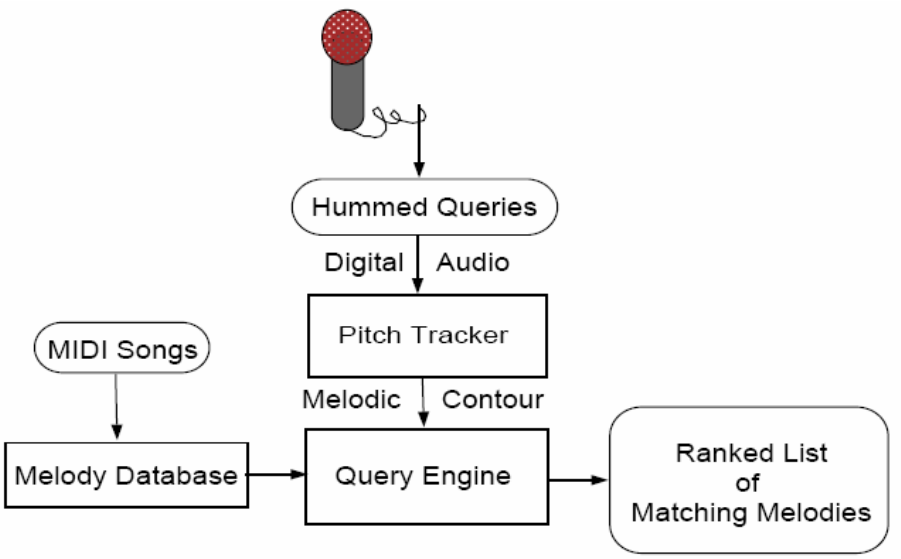

Figure 25
Proposed MIR System Diagram by Ghias et al.

Many systems have focused on the use of vocal input as a query mechanism. One such system by Rolland et al. incorporates the idea of hummed queries into their *Melodiscov* system, and suggests referring to query by humming with the acronym WYHIWYG, or "What You Hum is What You Get" [Rol00]. Also, the PROMS/MiDiLiB project is suited for vocal input due to its error handling for poorly-formed queries, and can even support polyphonic queries specified using multiple voice overlays [CEMS00, CKE01].

4.2 Query By Instrument/MIDI

In a similar fashion to query by humming, instruments can be used as a means of query input for MIR systems. The instrument used is normally a MIDI-enabled device, although the usage of any musical instrument is possible given that transcription of the audio recording is also possible [DR01]. The advantage of using MIDI as an input form, either from an instrument or a pre-recorded MIDI file, is that the query transcription process is simplified. One system that partially targets MIDI input is the recently proposed Muugle[13] system by Bosma et al. [BVW06]. Muugle is capable of using input from four different sources, three of which are MIDI-based: query by humming (non-MIDI), a MIDI file, a MIDI-enabled keyboard (or other device),

[13] http://pierement.zoo.cs.uu.nl/muugle/

or an online software keyboard which functions in the same manner as a real-world MIDI device. Additionally, the prototypical MIR system proposed by Takasu et al. uses a MIDI instrument as the primary method of capturing a user query, although they acknowledge the use of the "query by humming" paradigm as an equally viable alternative given a sufficient solution to the transcription of vocal recordings into a MIDI-based representation [TY99]. The main drawback of instrument-based approaches is that they require the user to be familiar with the musical instrument used for querying; additionally, users prefer searching to be fast, and playing back an instrument often takes slightly more time than casually singing or humming a tune.

4.3 Query By Note Placement/Score Notation

The ideal input paradigm for musical queries would be a fully featured musical query specified in a digital score format (see Section 2.1.3). This would allow a feature reduction algorithm to be applied to the data without the need for complex audio transcription, and would most certainly lead to improved search results. Despite these advantages, using musical score as a query interface is universally viewed as inappropriate for MIR systems, at least by default. The composition of musical phrases through score notation is considered complex, even to those who are familiar with many aspects of music, and the potential for errors in the query specification is also quite high. Furthermore, it is highly likely that users would become frustrated with the amount of time needed to fully specify a query. Still, some systems do allow queries to be manipulated using note placement, such as the Muugle system's software keyboard mode [BVW06].

4.4 Query By Rhythmic Gesture

Rhythm is normally incorporated as an aspect (feature) of music to be considered in any of the input paradigms outlined in Sections 4.1-4.3, rather than a separate and distinct paradigm itself. Still, research has been conducted to explore the possibility of queries formed using rhythmic data alone, which is usually captured by having users tap a rhythm. Chen and Chen outline a full MIR system framework based on musical matching using rhythmic features exclusively; they leave the mechanism of querying unspecified, however their model supports query by tapping as well as other input forms, such as MIDI files for rhythmic analysis [CC98].

In recent work by Kapur et al., a new and unique rhythmically-based query paradigm is introduced, query by *beat-boxing*, which allows a user (such as a DJ) to gutturally mimic the rhythmic texture of a musical phrase [KBT04]. Interestingly, this also unlocks the polyphonic dimension for the human voice in a way that makes sense; beat-boxing normally attempts to reproduce the percussive instruments of several musical parts at once, a task which is extremely difficult (if not impossible) to do reliably when providing pitch information vocally. Although test results are not given

in the paper, the potential for greater matching using multiple rhythmic tracks shows promise for the beat-boxing paradigm, and a possible reinvigoration of rhythmically-based MIR research.

5.0 Pattern Matching and Induction

Musical pattern *induction* involves the identification and extraction of patterns within musical pieces; it is widely considered by various authors that musical themes can found amongst musical patterns, and that users perceive music based on a thematic model of perception. This has led to several works attempting to identify musical themes by hand, perhaps the most notable of these by Barlow and Morgenstern[14] [BM48]. Musical patterns can be discrete, involving only parts of music, or they can be comprehensive and fully featured repetitions within a musical phrase. The advantage of identifying these patterns and storing them within a musical database is that an MIR system can search (match) amongst only the patterns that are thematically relevant, rather than the entire musical passages that contain them. This will to minimize the workload of searching algorithms while at the same time decreasing the likelihood of useless false matches when attempting to identify a musical theme. For reference, pattern *matching*, the process of gauging the similarity of two musical passages, is sometimes referred to as pattern *recognition* [CCI01, Rol01b].

5.1 What is a Theme?

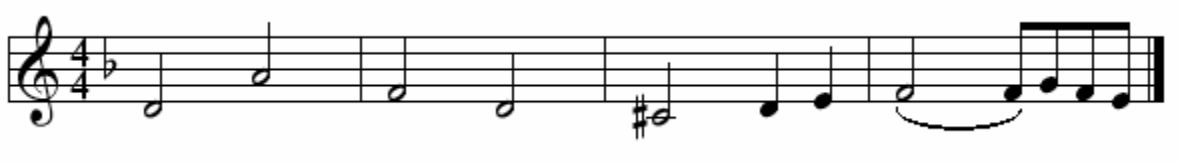

Figure 26
The main theme of Bach's "The Art of the Fugue", identified by hand[14]

A musical theme is a specific set of notes or features within a musical piece that carries an identifying property, where users recognize the set as representative of the piece as a whole. This dilution is itself a characteristic feature of a piece of music, though more difficult to identify than mere pitch or rhythmic features; themes are actually an aggregation of information from these domains. Informally, a theme is the "tune that is whistled" when a person attempts to remember and/or summarize the character of a piece, and there may be several themes within a single work, depending on how the piece is composed. Meek and Birmingham state that themes can occur anywhere within a piece; not just at the beginning, and that themes can occur in any voice, not just the upper voice of polyphonic music [MB01]. Furthermore, themes often interweave throughout a piece, repeating several times within a melody to serve as an anchor within a musical passage. For this reason, many researchers feel that a theme can be automatically extracted through the

[14] http://www.multimedialibrary.com/barlow/index.asp

identification of maximal repeating patterns within music; work by these authors' efforts are examined in the sections below.

Despite the widespread use of themes as a primary mechanism for identifying musical pieces, a practice dating back at least a thousand years [Bro80] and still prevalent today, some authors feel that offering full-text search over all of the notes within a musical database is preferable [DN00]. The debate as to the best approach is one of opinion; should non-thematic musical excerpts be searchable? If yes, then it is possible that a theme of one piece will occur as a passive and trivial phrase of another piece, thus potentially leading to mismatches. If no, then some searching power is lost within the MIR system. Of course, a system offering both options is likely to take advantage of the strengths of each approach to increase robustness.

5.2 What is a Musical Pattern?

As is common with most terms used by the MIR research community, the exact definition of a musical pattern is not entirely agreed upon [RG02]; the definition adopted by various authors usually supports their own work, and this serves as a domain-specific term rather than a general one. Rolland defines a *sequence* as a naturally-ordered set of data, and refers to musical patterns as *sequences* of music [Rol01a]. This definition is vague and tends to de-emphasize patterns such as rhythmic histograms that occur in musical phrases. Meudic defines a musical pattern as a "perceptible repetition in a musical piece" [Meu03], which focuses on the repetition of music within a self-contained piece. Meudic then, with co-author St. James, realizes that additional questions can be raised if one were to adopt his own definition; for instance, if a single or multiple part(s) of a polyphonic phrase repeat, does that constitute a musical pattern? Should the entire polyphonic context repeat in order for a pattern to be identified? [MS03]. In general, there are two ways to consider a pattern; one considers a pattern as something that repeats, whereas the other considers a pattern as some footprint of a musical passage that does not necessarily imply repetition at all. In practice, most prominent MIR researchers tend to define musical patterns as entities that identify repetition [Kru90, Lar03, LHC99, NO04], and thus further references in this document will regard patterns as recurring features of music.

It is important to distinguish musical patterns according to their perceptual significance [LJ83]. For example, the single note A_4 (440Hz) may occur multiple times within a single piece, yet very few MIR systems would identify this as a pattern, because singular notes are simply not significant or noteworthy repetitions to the average user. A common automatic method of judging which patterns are perceptually relevant is to assign weights to each pattern, where longer patterns are weighted more significantly amidst many potential criteria. This approach is not unlike individual note weights for determining accentuated note division (see Section3.2.3). After weights have been assigned, the heaviest patterns may be selected with some threshold, thus eliminating trivially short patterns that can occur

by chance [NO04]. The notion of perceptual significance also raises questions about what constitutes repetition, at least from a human perspective. Strict repetition within a piece can occur sparsely, and quite often a composer will change slightly the motifs used to make the piece sound more interesting; thus some room for pattern mismatch or flexibility should be taken into account [CCI01]. There are many possible augmentations that may still be identified as a pattern to the listener, such as musical diminution, ornamentation, note substitution, and even the backward playing of some passages [MLW02]. Thus, pattern induction algorithms should be designed with some leniency when comparing the repetition of differing passages within a piece, while at the same time ensuring that the results make sense from a human point of view.

5.2.1 A Word on Maximal Repeating Patterns (MRPs)

One of the common features sought out when searching for useful repetitions within a musical piece is the maximal repeating pattern (MRP). Koh and Yu define MRPs in the following (condensed) manner [KY01]:

- A *sequence* of notes (X) is a set of notes that occur one after another within some musical bounds. A musical piece (M) is an ordered sequence of n notes where n represents the number of notes within the piece.
- A *repeating pattern* is a sequence of notes (X) within a musical piece (M) such that X is a subsequence of M and occurs at least y times within M (overlapping allowed), where y is some minimum threshold.
- A ***maximal repeating pattern*** (X) in M satisfies the following:
    - X is a repeating pattern within M
    - There does not exist any repeating pattern X' in M where X is a subsequence of X' and X' occurs the same number of times in M as X does

A similar definition for MRPs is given in [LHC99], though the term used for maximal repeating pattern is "non-trivial". To view this plainly, the set of maximal repeating patterns within a musical piece will contain the longest patterns that meet a minimum threshold of repetition within a musical piece. The usefulness of identifying and storing MRPs within a database relates to the identification of themes; since themes tend to be the longest repeating sections of music, identifying all of the maximal repeating patterns within a passage of music should result in a set that contains a musical theme, or at least a significant part of one. Thus, by reducing the set of information to specific, choice patterns that are perceptible and relevant to themes, misleading and useless information is discarded, making the search process more focused and potentially more relevant [MB01].

5.3 Pattern Matching Approaches

The difficulty of comparing two musical patterns against each other depends greatly on the precision of matching similarity; that is, how close must the patterns be to each other? There are two general responses to this question:

- **Exact Matching** – In exact matching, two musical patterns must be identical in all aspects of the pattern, regardless of how the pattern is formed (multiple sequences, histogrammatical, etc.). Few existing MIR systems rely on exact matching between patterns, especially when matching a query to a database record, since human beings often perceive slightly-different patterns as a match.
- **Partial Matching** – By far the most commonly used paradigm, partial matching allows patterns to match each other as long as a certain amount of similarity exists between the two. This is widely considered to be a better approach, since most composers tend to use variations and augmentations on a main theme when constructing a piece as a whole. Thus, being sensitive to the small variance that can occur between patterns can increase matching accuracy greatly according to the human perception of music. Figure 27 shows an example of two passages that match partially since one is derived from the other, even though visually and mathematically the match might not be obvious. Thus, designing algorithms to induce and recognize these patterns can be difficult.

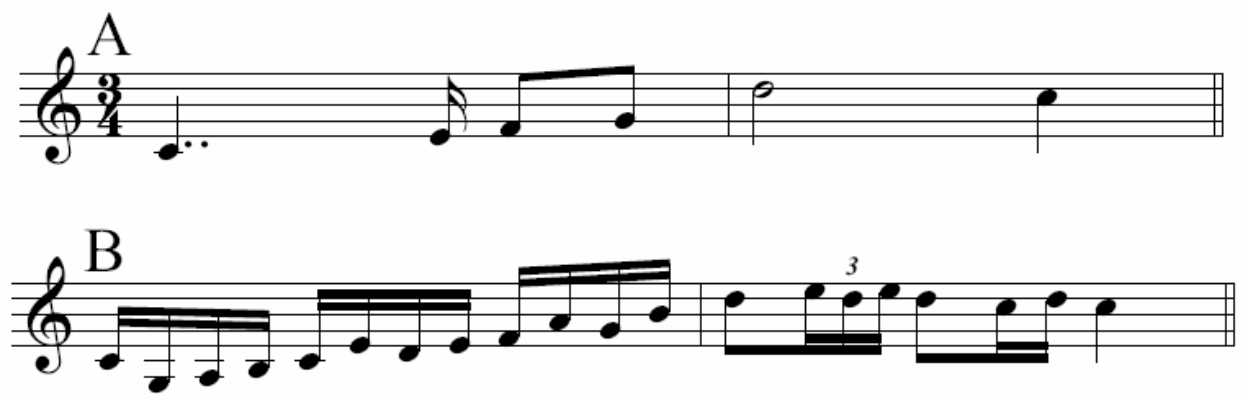

Figure 27
Pattern B is an embellishment of pattern A [MLW02]

In order to gauge similarity between two musical patterns, one must ask the question "*how* can we identify pattern similarity?". The similarity models below outline some of the approaches used when comparing music to itself.

5.3.1 {δ,γ}-Approximation

When dealing with string-based representations, δ-approximate and γ-approximate matching are ways to judge the similarity of two musical string patterns of equal length, both of which (δ,γ), can be determined separately or together. Consider three pitch interval strings:

- String A = 4, 3, 2, 4
- String B = 5, 3, 1, 5
- String C = 4, 3, 0, 4

The value δ in δ-approximate matching determines the maximum difference between two specific symbols in a musical string; that is, for each individual comparison in two strings of equal length, the values at any particular position must not be more different than δ. For example, with δ=1 for δ-approximation, strings A and B are considered similar, since the difference between their corresponding symbols is 1, 0, 1, 1 respectively. Since no pairing has more than 1 difference, a successful match is made. Conversely, while strings A and C seem similar visually, they are not δ-approximate for δ=1, since the third symbols of the strings have a difference of 2.

The value γ in γ-approximate matching determines the maximum total difference between two musical strings of equal length; that is, when calculating the individual comparisons in two strings, the sum of the resulting differences will not exceed γ. For example, with γ=2, strings A and B are NOT γ-approximate, since the differences between their corresponding symbols are 1, 0, 1, 1 which means a total difference of 3. Conversely, strings A and C are γ-approximate with γ=2, since their corresponding differences are 0, 0, 2, 0 which gives a total difference of 2, within tolerance.

{δ,γ}-approximate matching is the union of the rules of δ-approximation and γ-approximation; that is, the criteria for both must be met for values of δ and γ. For example, with δ = 2 and γ = 2, strings A and B are not {δ,γ}-approximate; while both strings meet the δ criteria, they are too different totally to meet the criteria for γ. Conversely, strings B and C are {δ,γ}-approximate; their differences are 1, 0, 1, 0, which means they are within both tolerances for δ and γ.

δ-approximate and γ-approximate matching offers advantages for gauging perceptual similarity; musical patterns are often slightly-modified versions of each other, and thus δ-approximate matching allows for slight variation and ornamentation that does not deviate too far from some original pattern string. On the other hand, γ-approximate matching allows a total maximum deviation to be set, so that too much ornamentation or augmentation from the original pattern can be identified as a unique pattern itself. These approaches are used in a variety of papers and by a variety of authors; some classic examples can be found in [CIMR+02, ILMP00, CCIM+99].

5.3.2 Matching with Don't Cares

Matching with binary don't cares, or simply "don't cares", is a tolerant string-matching method that allows some symbols to be matched to multiple targets, rather than follow the usual 1:1 mapping of symbol similarity. For example, consider the following step-leap contour strings (see Section 2.3.3.2):

- String A = +S, R, –S, –L, +L
- String B = +S, R, +S, –S, +L

When comparing these two strings, one would normally use a 1:1 binary comparison scheme, such that +S=+S, R=R, –S=–S, +L=+L, and –L=–L. However, don't care symbols can match multiple targets; two commonly-used don't care symbols are * and #, which add the following matching relationships: *=+S, *=+L, #=-S, #=-L. In other words, the * don't care specifies no preference between being matched to an upward step or leap, with the # stating the same for a downward pitch interval. The reasons for allowing this are numerous. For example, a query mechanism might determine that softly sung notes from a human user may indicate poor knowledge of a portion of a musical theme; thus, those notes can be replaced by don't cares which allow fuzzier and more tolerant matching for parts that a user is not entire certain of. Matching strings with don't cares is therefore an example of exact matching rather than partial matching; this is counter-intuitive because using don't cares seems to imply that only a part of a musical passage will be accurately matched. A better description, however, would be that the entire string is accurately and exactly matched due to a broader matching mechanism.

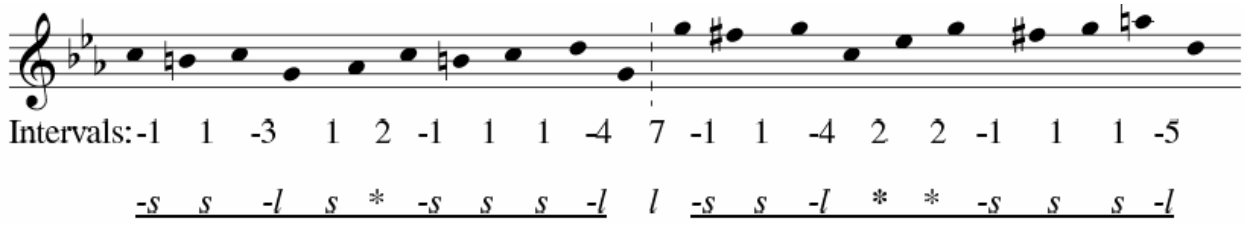

Figure 28
Passage consisting of two similar passages making a "repetition with a hole" [CCIM+05]

The use of don't cares is normally used with, though not limited to, step-leap contour representations. An example of this can be seen in Cambouropoulos et al., who uses the aforementioned don't care symbols * and # [CCIM+05]. Their approach finds all maximal repeating patterns, and uses don't cares to discover an additional entity called a "repetition with a hole". This is defined as follows: If two repeating patterns *a* and *b* are separated by exactly one interval, then that interval is given an appropriate don't care symbol (# or *). We define the entire concatenated string **w** = ***a*** M **{#,*}** M ***b*** where M represents string concatenation. If **w** matches another sequence within the musical passage, then it is considered a repeating pattern "with a hole", and added to the list of repeating patterns (it is likely an MRP, but not always). Figure 28 shows two patterns that form a repetition with a hole.

5.3.3 Searching with Gaps

Searching with gaps can be defined as finding patterns where trivial symbols separate the symbols of a pattern; in other words, the pattern can be broken up amongst non-pattern sequences symbols as well, though still appearing in sequential order. Consider a query and a musical string representing pitch values of a musical theme, such as the following:

- **Text**: 50, 57, 53, 50, 49, 50, 52, 53, 55, 53, 52
- **Query**: 50, 53, 49, 53

From simple observation, the query does not exist *verbatim* anywhere within the text. However, it may be possible to identify the sequential symbols of the query if gaps are allowed. Consider Figure 29:

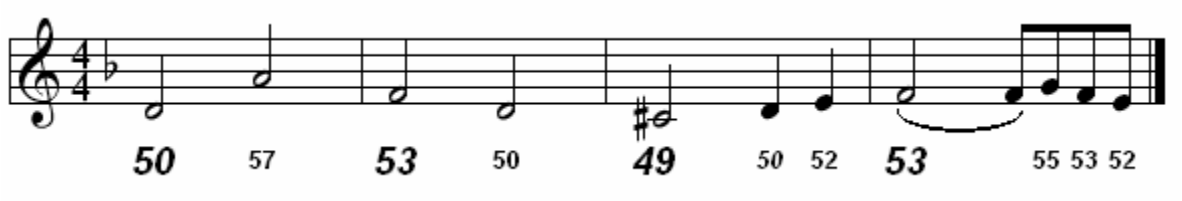

Figure 29
Absolute MIDI Pitch Values of Bach's "The Art of the Fugue"

By allowing gaps between the symbols of the query, additional patterns can be matched that may be semantically relevant. In the case of Figure 29, a match for the query was found when this approach was taken. The problem of searching with gaps is similar to the problem of finding "repetitions with a hole", as outlined earlier in Section 5.3.2; the main difference here is that each space between non-trivial symbols can be more than one symbol long. Thus, searching with gaps is similar to searching for patterns with contiguous runs of *don't care* symbols within a text. This can be especially useful for searching polyphonic music represented in a single string, where less emphasis on a good voice division algorithm can be adopted for increased flexibility by allowing gaps in the searching process.

Use of this approach is explored by Antoniou et al., who use finite automata to represent systems that can identify patterns with gaps [AHIM+06]. Additionally, gap searching is heavily explored by Crochemore et al., who outline a number of algorithms to achieve matching in several specific gap situations [CIMR+02]. Among these algorithms, the following specific gap searching problems are explored:

- α-bounded gap searching, where α represents the maximum gap length that can occur between two matched symbols
- α-strict-bounded gap searching, where gaps between symbols must be exactly α in length

- ε-bounded gap searching, where ε represents the maximum gap length in total than can divide the sequence of non-trivial pattern symbols
- a number of algorithms that combine the previous three approaches with $\{\delta,\gamma\}$-approximate matching (see Section 5.3.1).

5.3.4 Levenshtein (Edit) Distance

One of the most common ways to judge string similarity is the use of Levenshtein distance, or edit distance [Lev66]. Edit distance is defined as the minimum number of editing operations that must be performed to make the strings identical [HD02]; classical Levenshtein distance outlines three editing operations:

- **Insertion** – A symbol is inserted into a string, making a string of length n increase to a length of n+1
- **Deletion** – A symbol is deleted from a string, making a string of length n decrease to a length of n-1
- **Replacement/Substitution** – A symbol is replaced with an existing symbol in a string, resulting in no length increase or decrease

As an example, consider the words BARN and ARMS. The Levenshtein distance between these two words is 3; that is, it would take three edit operations to transform BARN into ARMS:

- Operation #1: DELETE B FROM BARN ➔ ARN
- Operation #2: REPLACE N IN ARN WITH M ➔ ARM
- Operation #3: INSERT S INTO ARM ➔ ARMS

While the term edit distance normally refers to the 3-operation Levenshtein model, other operations can be used to determine edit distance as well. Mongeau and Sankoff extend Levenshtein's 3 basic operations by an additional two operations: *fragmentation* allows a symbol to be broken up into multiple symbols, and *consolidation* allows multiple symbols to be replaced by a single symbol [MS90]. Edit distance operations can also be associated with weights; the example above shows an edit cost of 3 to convert BARN into ARMS, which assumes that each of the 3 possible edit operations incur a cost of 1 respectively. However, certain edit costs can be altered by associating uneven weights to each, allowing for more complex edit distances to be considered. This suits approaches such as the work of Hu et al., who propose the comparison of musical patterns based on the (weighted) probability that one pattern was derived from the other, which naturally invites the use of weighted edit distance as a measurement [HD02].

The calculation of edit distance is normally done using a correlative matrix with the two patterns being compared down each axis. The operations allowed for the edit model form a pattern such that values in the matrix are selected to minimize

the growing edit distance on the downward diagonal of the matrix. For example, in Figure 30 the 5 operations allowed form a pattern where the lowest value of *previous edit distance* + *edit cost* is selected for the diagonal value.

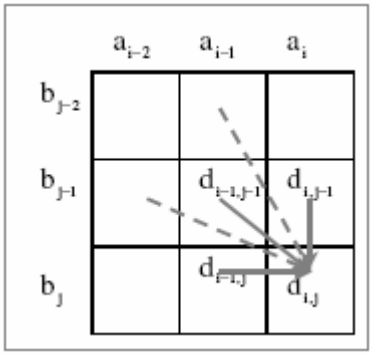

Figure 30
Calculation pattern of the 5 edit operations allowed by Mongeau and Sankoff

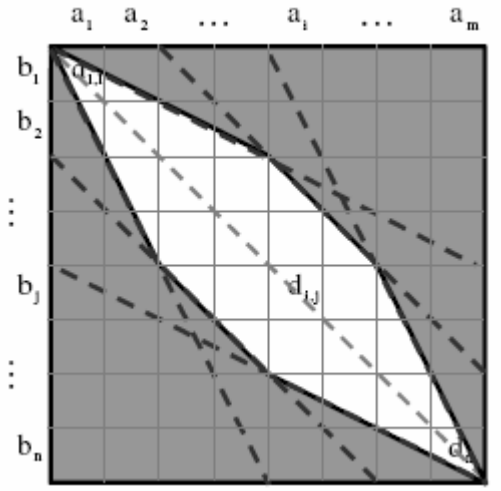

Figure 31
Windowed constraint for use in edit distance calculation [HD02]

Since the entire correlative matrix must be considered to effectively determine the edit distance between two strings, calculation can be quite inefficient when dealing with the comparison of larger strings. Many authors use a windowed constraint approach, where the calculation of very far edits are ignored, tangibly improving efficiency. For example, Figure 31 graphically depicts the window used by Hu and Dannenberg, who also slide the window along the string text to compare patterns at each available position in the musical phrase (Figure 32). The grey areas of the matrix are filled with a maximum distance tolerance value so that the calculation of these areas is not required and close alignment is preserved; expensive and unlikely edits will cause the calculation of the matrix to halt.

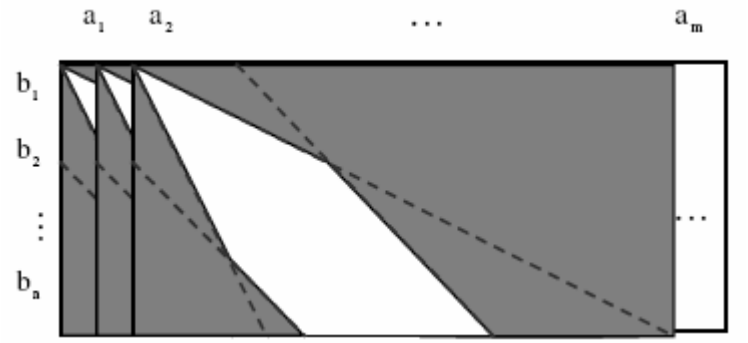

Figure 32
Sliding constraint windows used to calculate edit distances at multiple positions [HD02]

5.3.5 Smooth Pitch Contours

Most approaches toward MIR systems focus on string processing, and while this has led to fast searching algorithms as was intended, recall accuracy has room for improvement. The comparison of smooth pitch contours seems to be a growing area of interest in recent MIR research, as some authors begin to focus on precision, even while sacrificing speed in the process [AMW05]. The problem with most significantly reduced representations (like strings) is that too much information is lost during musical transcription processes, as algorithms tend to discard data that is complex to deal with, and yet still potentially important. In the general sense, a pitch string is a very inaccurate sequence when compared to the thousands of frequency samples used to create the string originally. Thus, it is better to match a smoother contour, where multiple pitch frequency estimations are used for each note; a compromise between the *1:1 note vs. symbol* relationship in strings, and the *VAST:1 information overload* of sampled audio.

Mazzoni and Dannenberg's early attempt at matching smooth contours provided a basis for further research into their use [MD01]. In their approach, MIDI files are broken down into a finer granularity than common strings; the pitch frequency occurring at every 100ms of the musical phrase was recorded, thus forming a contour. While their approach focused on MIDI files for source data, queries into the system followed the query-by-humming paradigm (see Section 4.1), thus demonstrating their efforts on sampled audio as well. The query is converted into a smooth, fine-grained contour, and then overlaid on top of each target melody's contour. A distance is measured, which results in a ranking, and the smallest distance wins for all comparisons in the database. While this resulted in slower processing times, accuracy was successfully increased.

In recent work by Adams et al., the use of smooth contours is augmented by iterative deepening to increase processing efficiency while maintaining accuracy [AMW05]. The authors recognize that when comparing smooth pitch contours, time information is inherent in the approach since the contour is plotted against time. However, queries may not match (or start at) the exact time for the stored smooth

contour. Thus, some alignment must be done, and iterative deepening is used to achieve dynamic alignment (called *Dynamic Time Warping*) during the search process. The query is compared to all theme contours, first with short distances. If initial comparisons are too far apart, the remainder of the contour will not be compared any further with that theme, and the theme is disqualified. Alignment occurs when enough iterations (3 to be exact in their experiments) occur to find the best-fit theme for the query contour. The results are promising, with a claimed increase in efficiency by a factor between 25 and 40 over previous methods.

5.3.6 Comparison of Histograms

The extraction of musical information through histograms has been sparsely explored by a few MIR systems, using the information gathered to serve both primary and secondary functions. As is the case with most musical extraction methods, the histograms are usually formed from pitch information, where intervals or exact pitch values are grouped logically within some time interval. The major advantage of using histograms as a means for comparing two musical phrases is speed and simplicity; most algorithms for information gathering and comparison are easy to design, and can scale well to large databases of musical themes. Naturally, this approach can lead to inaccuracies in matching, since the data gathered is purely statistical, and normally lacks a full time-ordering which perpetuates false matches.

Tzanetakis et al. make use of pitch histograms in a novel, secondary way for the realization of an effective MIR system [TEC02]. Their approach analyzes histograms formed from musical pieces in an attempt to automatically classify the piece into one of five target genres {Irish folk, Classical, Jazz, Rock, Electronica} based on prior musical knowledge gained through machine learning algorithms on existing categorized pieces. While their success was poor for some genres, the approach could be refined to assist an MIR system that considers genre classification when conducting searches, or when attempting to segment a melodic line (see Section 5.4.11).

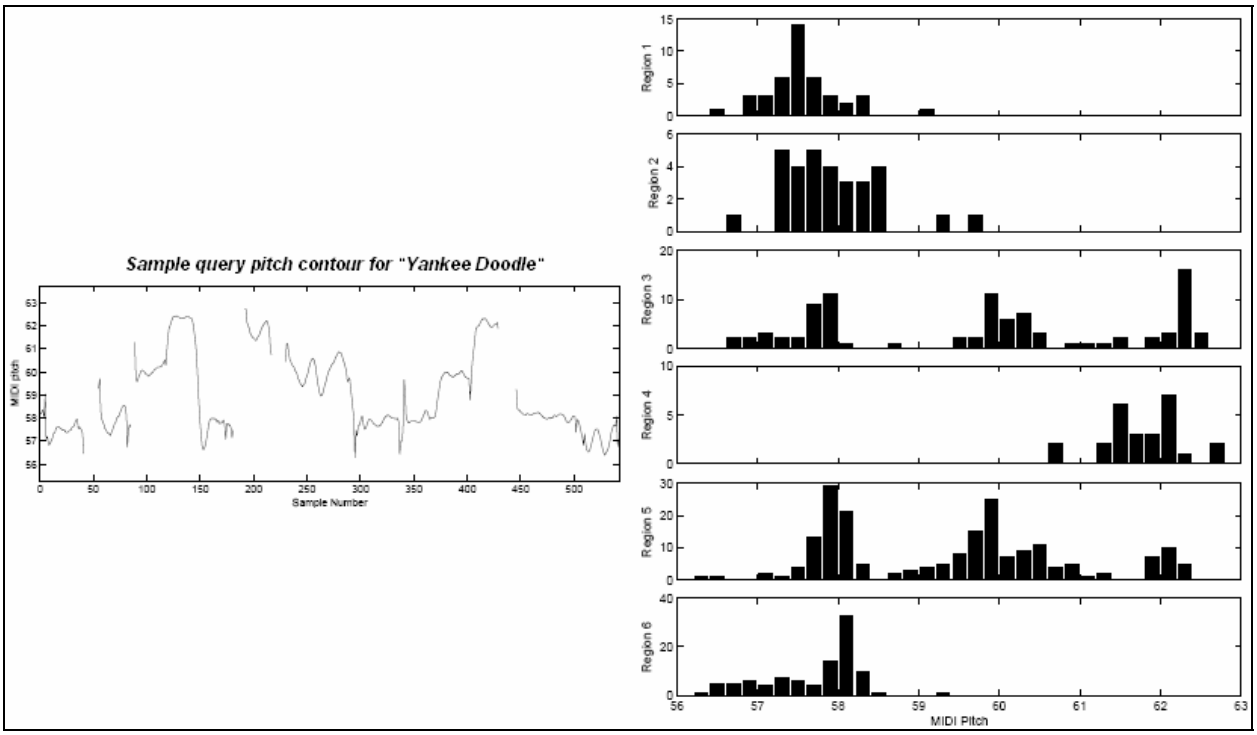

Figure 33
Histogram regions calculated in [Ada04], for a query of "Yankee Doodle"

Other work by Adams et al. attempted to compare three approaches for matching musical information: string sequences of notes, smooth contours, and pitch histograms [Ada04]. The histogram approach took a set of histograms from queries and melodies; this was done by breaking up each musical phrase into regions, and taking a histogram representing the frequency of occurrence of pitches within that region (see example in Figure 33). The target and query were then matched based on a simple distance measurement between their corresponding histograms. After analysis of the three methods used, the authors found that the histogram approach led to a good balance between the accuracy of smooth contours and the efficiency of string-based approaches.

5.4 Pattern Induction Approaches

The problem of extracting significant repeating patterns from a musical piece has been heavily explored by several authors. Most approaches attempt to find patterns in strings of symbols, however each approach ultimately relies on the underlying representation used for the music. The fundamental approaches below (and some noteworthy and illustrative algorithms) outline some of the devices used for identifying patterns in music; many of these devices can and have been used together simultaneously, both in research and also in real-world MIR systems. Thus, the sections below attempt to identify some building blocks of the many pattern induction algorithms available to date. It should also be noted that many of these

approaches can be used for *matching* a query to a pattern within a database, since the fundamental idea is very similar: given a text and a pattern, find all occurrences of that pattern within the text [CCIM+99].

5.4.1 Use of Trees/Tries/Lattices

The organization of data into trees has long been an effective approach for a vast number of searching algorithms. Many pattern induction techniques target these structures due to the efficient search times associated with them; this is sometimes a trade-off with space complexity, since large tree-like indices can have high storage costs, depending on the type of tree used.

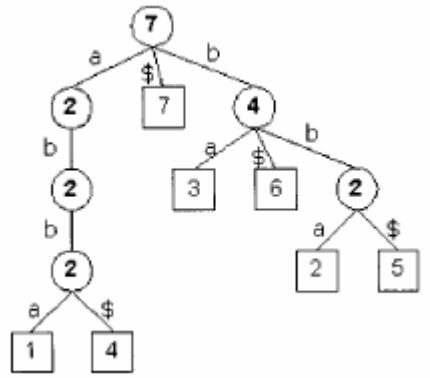

Figure 34
Example of suffix tree of a feature string S = "ABBABB" [HLC01]

Investigation into tree and lattice usage for MIR systems is widespread. An excellent and well-cited pair of papers explores, among other things, the use of suffix trie's for the purpose of pattern identification [HLC01, LHC99]. These authors use a type of suffix tree (called RP-tree) to store substrings, and then join suffixes together based on positional information, while discarding all trivial patterns found during the process. The process of joining strings together continues until only MRPs are left. Verbeurgt et al. use suffix trees to identify musical patterns, not for the purpose of musical information retrieval but instead for use in music generation [VDF04]. They induce patterns by constructing a suffix tree and extracting right-maximal string sub-sequences of pitch intervals. The resultant strings are statistically analyzed to provide probabilistic input for musical composition via Markov chains.

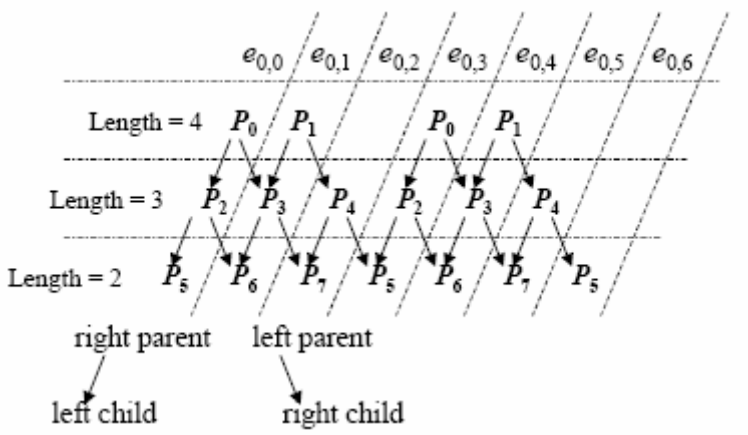

Figure 35
Meek and Birmingham's lattice for the first phrase of Mozart's "Symphony no. 40"

In another excellent paper by Meek and Birmingham, the Melodic Motive Extractor (MME) algorithm uses a lattice structure very similar to a tree, but allowing multiple parents for each node (see Figure 35) [MB01]. Their approach uses MIDI files, ignoring any that are malformed, and extracts only the top voice of all multi-voiced instruments within each MIDI channel into a pitch interval contour. Many other features are calculated, and patterns are organized by weight; a large factor of weight is the frequency of occurrence of a particular pattern. Their approach achieves highly successful theme extraction rates, ranging between 85% and 100% effectiveness in trials. The authors also note that their approach is attractive because it was formulated with little musical knowledge; that is, the authors did not have to grasp the more complex concepts of musical theory to attain a desirable result.

5.4.2 Use of *n*-grams

Musical n-grams are equal-length bits of information, usually derived from a single feature string such as a pitch interval sequence. Thus, a complete musical piece represented as a string can be divided into *n*-grams of length *n*, and stored as such with no symbol loss from the original string(s) used to create the *n*-grams. The extraction of *n*-grams is actually a deviation from true pattern induction; the majority of *n*-grams extracted will not represent any perceptually significant repeating pattern, if the n-gram even repeats at all. The use of n-grams instead (usually) aims to provide full-text access to musical pieces, rather than access at a thematic level only. While this requires more work for algorithms to search an entire database of music, advances in musical string searching can alleviate the real-world burden incurred when dealing with many *n*-grams. One area which has been not been explored adequately is the possible storage of n-grams in tree structures, which could further improve searching efficiency.

Downie and Nelson break down monophonic musical passages into a series of pitch intervals of length n, to obtain an exhaustive *n-grammed* representation of musical pieces [DN00]. In their work, several *n*-gram databases were created and compared to study the effects of varying lengths of *n*, as well as varying alphabet sizes for the *n*-gram strings. They found that, with adequate alphabetic cardinality and sizes for *n*, *n*-grammed approaches can be effective for allowing full-text retrieval within musical passages with reasonable efficiency. Still, their approach is quite limited since it can only handle monophonic music; this either limits the potential set of music that can be analyzed, or relegates the complexity of dealing with polyphonic music to algorithms that can extract monophonic passages first in order to provide properly-formed input.

The approach by Downie and Nelson has been improved by Doraisamy and Rüger, who attempt to use the storage of musical *n*-grams for the purpose of polyphonic full-text searching [DR01]. They use a sliding window to capture all polyphonic notes within a frame, and for each slide they calculate all of the possible combinations of time-ordered polyphonic notes, each one being stored as a new *n*-gram. Like Downie and Nelson, the authors use upper-case letters to represent (quantize) increasing pitch intervals, and lower-case letters to represent interval drops. The results obtained from this approach left some to be desired; recall accuracy is reduced compared to the monophonic *n*-gram approach, and the time and space complexity of searching the vast sets of n-grams representing all possible passages through each polyphonic musical phrase was considerable. Also, since most users search pieces based on themes and repetitive passages, it is natural to question the real utility of *n*-grammed systems for general use, when compared to other proven and effective approaches.

5.4.3 Bit Parallelism

The main idea of bit parallelism is to pack values that require processing into a machine word, to take advantage of the fast update operations inherent with modern computer hardware, such as bit shifting operations. Sometimes these approaches require the storage of bit operation results in tables, for later analysis.

Cambouropoulos et al. provide an approach to δ-approximate and $\{\delta,\gamma\}$-approximate string searching that makes use of bit parallelism [CCIM+99]. The approach works as follows: for a given musical string of note events, patterns are identified by comparing the string against a text in a sliding fashion. This results in a series of "delta states", such that each state is a binary sequence where a 1 appears if two symbols are at most δ apart, and a 0 otherwise. The delta states are then packed into machine words, and a series of binary operations (SHIFT, AND, OR, etc..) are applied to determine the frequency of the string in the text. The entire algorithm, called SHIFT-PLUS, uses dynamic programming and tables to remember past values, and has been widely accepted as a successful and efficient way to induce $\{\delta,\gamma\}$-approximate patterns in musical strings.

Other work by Koh and Yu attempts to improve on efforts by Liu et al [KY01, LHC99]. They create a bit index table for each musical passage in a large MIR database. The initial number of entries in the table is equal to the number of distinct notes in the passage. For each entry, a bit sequence is stored that contains a 1 value for all string positions where the corresponding distinct note appears, and a zero otherwise (see Figure 36, noting that the least-significant bit is considered as bit number one):

Note	Bit index sequence
Re	10000000
Mi	00001000
Fa	01110000
So	00000111

Figure 36
The bit index table for the length-8 string SoSoSoMiFaFaFaRe. Adapted from [KY01]

The authors then devise an algorithm which uses a binary string join approach: patterns of length 1 (bit index sequences which contain multiple 1 values) are identified as a repeating patterns, and each of these are merged with all other repeating patterns that have repeating patterns in adjacent bit positions. Each merged result is tested to see if it is also repeating. If so, it is further merged with repeating patterns in the same manner, until eventually no additional patterns can be merged. A second phase is then applied that filters out MRPs, leading to significant increases in both time and space complexity when compared to [LHC99].

One possible idea that has not been fully explored is the use of bit parallelism to process very simple reduced pitch contours similar to step-leap and U,D,S (see Sections 2.3.3.2 and 2.3.3.1). If the alphabet could be reduced to a size of two, say a 1 if the pitch raises or stays the same, and a zero otherwise, the potential for fast pattern induction and searching would be great. This would require significant testing however, since the information reduction imposed by such a small contour alphabet could render the matching process too inaccurate to be of real-world use.

5.4.4 Application of Heuristics

This section makes casual mention of notable papers using heuristic algorithms, since the papers listed here were only partially investigated by this literature survey. Several algorithms presented by Crochemore et al. make use of heuristics for exact and/or partial matching with $\{\delta,\gamma\}$-approximation [CILP+02]; they apply two occurrence and substring heuristics to allow skipping of characters that have already been visited when matching strings, to reduce the number of pattern comparisons. Their approach uses a modified Boyer-Moore algorithm, suffix tries and subword

graphs, eventually realizing a speed increase over the bit parallel approach presented in [CCIM+99].

Takasu et al. divide musical phrases partly based on a simple heuristic where phrases are bound by musical rests appearing in MIDI files [TY99]. This adds an additional induction task, since there is no explicit way of expressing a rest within the MIDI file format; however, once rests have been identified the task of segmenting the music is made simpler. While some musical themes and repeating patterns do contain rests, dividing music this way is a promising idea that could use further exploration, since the computational burden of sliding windows could be significantly reduced by intelligently selecting passages to test for repetition.

5.4.5 Dynamic Programming

Since pattern matching often requires the memory of past results to form new matches, the process of pattern induction routinely makes use of dynamic programming. Normally, this is coupled with other approaches such as bit parallelism or correlative matrices (Sections 5.4.3 and 5.4.10), and is the de facto approach for algorithms that induce patterns based on edit distance measurements (refer to Section 5.3.4). A series of algorithms by Crochemore et al. make heavy use of dynamic programming to accomplish relaxed matching for finding patterns in strings with gaps [CIMR+02]. More recently, an algorithm by Cambouropoulos et al. uses dynamic programming with matrices to resolve the problem of finding patterns where don't care symbols are present, thus improving the simplicity of finding repetitions with a hole (see Section 5.3.2) [CCIM+05]. Other works that employ dynamic programming for the purpose of pattern induction are vast, including [ABSW04, Cam06, HCL98, LWC05, MS90, PK02, Rol98, Bak97, Tem01].

5.4.6 Multidimensional Projection

In a multidimensional projection approach, musical information is stored in a *k*-dimensional Euclidian space, where *k* represents the number of single musical features to be extracted from the musical source. Patterns are usually not defined as exactly repeating sequences, but rather groupings of notes within the space that satisfy a certain distance criteria to other note groups; these groups can be identified as a cluster, and any cluster with more than one note would be considered a repeating pattern against which a query can be compared. This provides some advantages:

- **Overcomes some limitations of string searching.** For example, this approach allows easier searching with gaps, since notes exist discretely in the *k*-dimensional space rather than sequentially in a string. Also, multidimensional projection is particularly suited for polyphony, as all notes can be plotted in

the same *k*-dimensional space, whereas multiple strings must be used for a full polyphonic representation.

- **Extensible and flexible.** Allows the creation of general *k*-dimensional algorithms that can handle an arbitrary number of musical features occurring on *k* axes.
- **Data reduction for targeting of specific features is easy.** The removal of dimensions from the multidimensional set can be done easily, allowing a reduction on data before algorithmic processing begins, to increase both accuracy and efficiency.

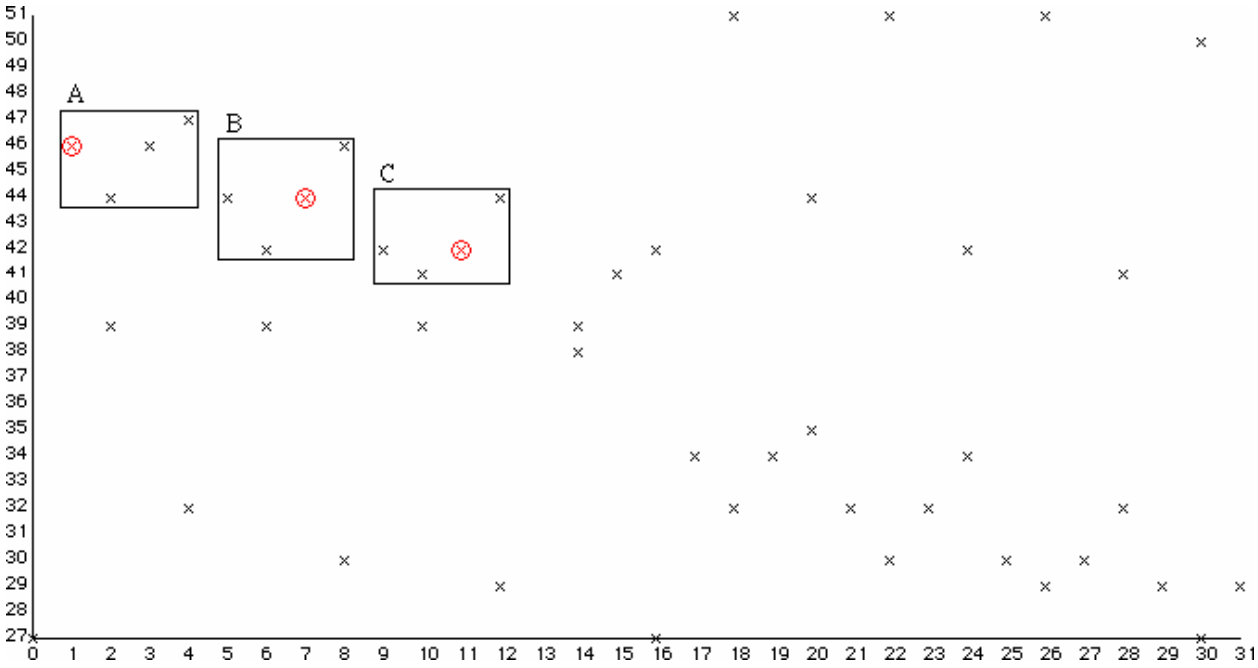

Figure 37
A sample 2D graph of pitch and onset time adapted from [MLW02]. Query notes are circled red. Boxes A-C represent points close enough to be considered as "similar" to query.

By plotting the musical note values discretely, queries can be converted to *k*-space points and tested for collisions (or some measure of distance) with original song points as a means of determining similarity between the query and the original music. Furthermore, databases can be organized in a couple of useful ways; one large *k*-dimensional space can store the plotted values of all the songs in the database, or each song can store it's own points in a separate space for that song alone. The former solution is usually a better fit, since searching for matches can occur very quickly due to the need to perform a search only once; of course, some mechanism to deal with collisions of notes from different songs in the same point of space would be needed, similar to the considerations of hash functions.

Meredith et al. introduce two algorithms that operate generally on k-dimensional datasets of musical information [MLW02]. They offer a slightly different definition of a maximal repeating pattern, which instead refers to the pattern with

the maximum length that can be converted to another pattern within the *k*D space (call a *maximum translatable pattern*, or MTP). The algorithms given are:

- **SIA**: Computes all maximal repeating patterns (calculates MTPs) within music based on *k* dimensions of data and different translation vectors. The algorithm achieves reasonable efficiency: $O(n^2)$.
- **SIATEC**: Computes classes of translationally-equivalent MTP classes within the dataset. Less efficient $O(n^3)$, but has a reasonable average case.

Other work by Shin et al. plots values in a 2-dimensional space defined by average length and pitch variation. They radii for all points in the graphical space, to represent a maximum difference to the original pattern, and then test queries using a distance measure against the clustered regions within the space (see Figure 38).

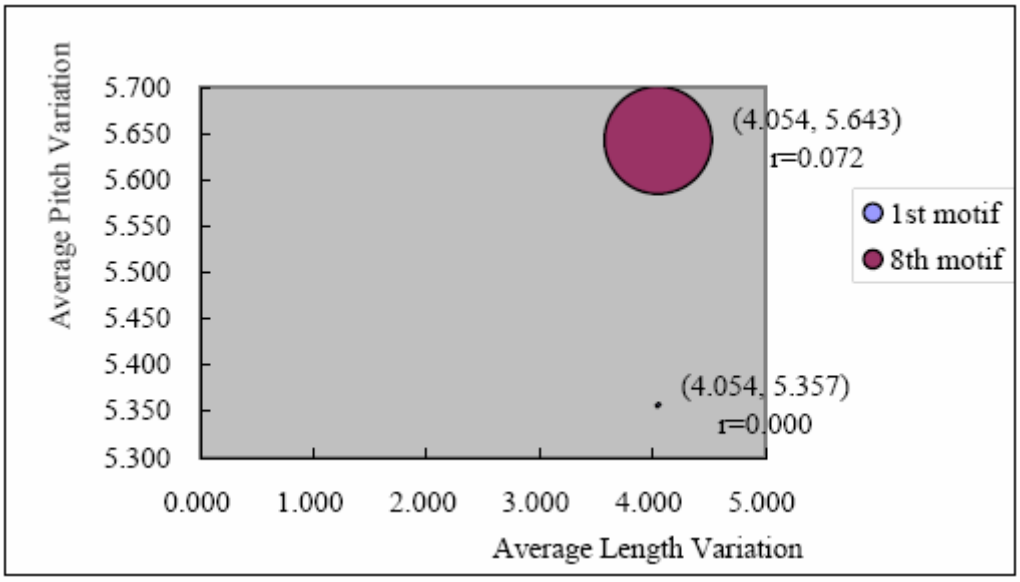

Figure 38
2D space with radial clusters for testing query note values, from [SKKK03]

5.4.7 String Joins

String joins, normally used in conjunction with suffix trees and tries, are a technique used to find maximal repeating patterns by first finding smaller repeating patterns and then joining them together based on positional information. Eventually, these string joins will result in the construction of maximal repeating patterns, and this occurs quickly since the size of the joined repeating patterns increases exponentially. For example, this approach is explored in [LHC99], and later in [HLC01] (see discussion of 5.4.1 for details).

5.4.8 Use of Graphs

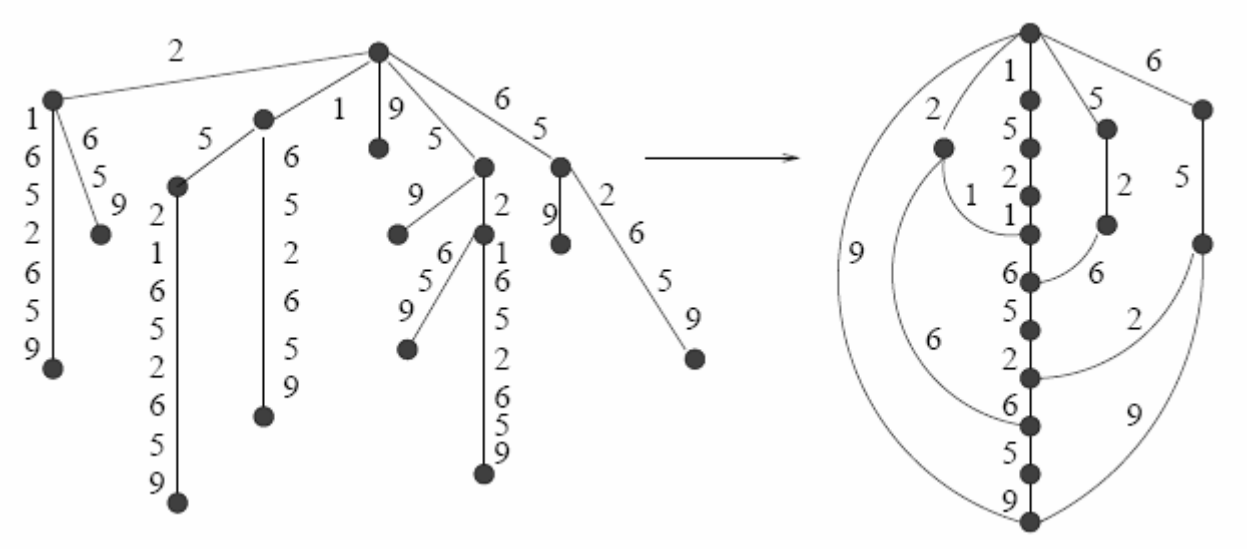

Figure 39
A suffix tree (left) and subword graph (right) for the word *w* = 1521652659 from [CILP+02]

Graphs can be used to form relationships between notes or pattern weights, where edges are placed based on a measure of some distance/similarity between the musical nodes. After such a graph is constructed, processing algorithms can take advantage of known practices in graph theory to analyze the relationships that occur in musical passages, leading to pattern discovery. This approach is used in an algorithm by Rolland called FlExPat, which stands for the Flexible Extraction of Sequential Patterns [Rol01a, Rol98]. The algorithm, which is widely recognized throughout MIR literature, as well as deployed in several real-world MIR systems, extracts sequential patterns that can be defined as "a set of segments from a sequential database which share a significant degree of resemblance" [RG02].

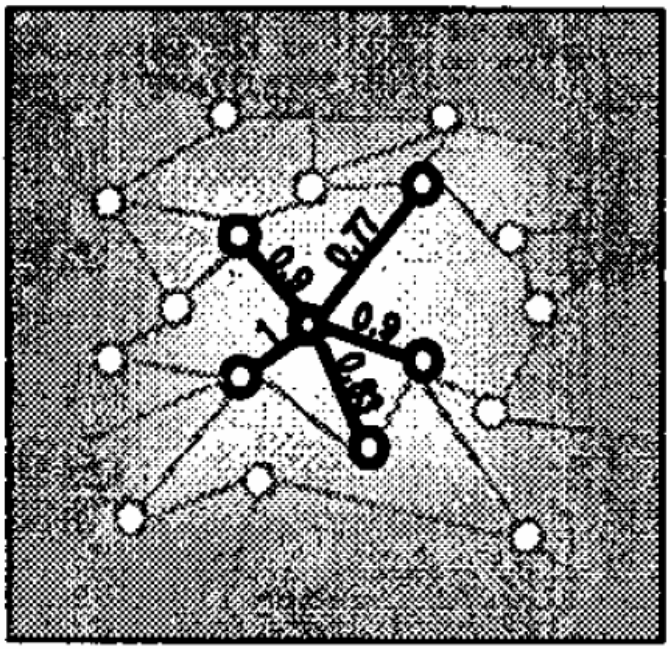

Figure 40
An example star-type pattern extracted from an equipollence graph [Rol01a]

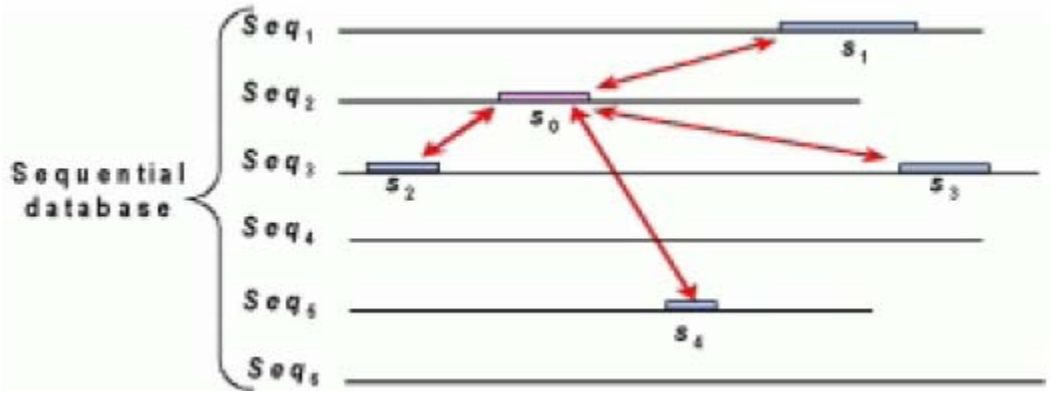

Figure 41
Another illustration of the notion of a star-type pattern. Segment s0 (called the star's centre) is equipollent to all other segments in the pattern [RG02]

The algorithm builds an equipollence graph where musical sequences are nodes, and two nodes are equipollent if they are sufficiently similar. The identification of star-type sub-graphs of sufficient similarity leads to the extraction and storage of patterns in a database or index; each star-type sub-graph is considered a *sequential pattern* (or just *pattern*), consisting of one musical string segment which is equipollent to all the other segments in the pattern (called the centre). These constructs are illustrated in Figure 40 and Figure 41.

5.4.9 Construction of Automata

The construction of automata can help to both induce patterns and verify queries. Recent work by Antoniou et al. focuses on the creation of a number of automata for the purpose of identifying repeating patterns [AHIM+06].

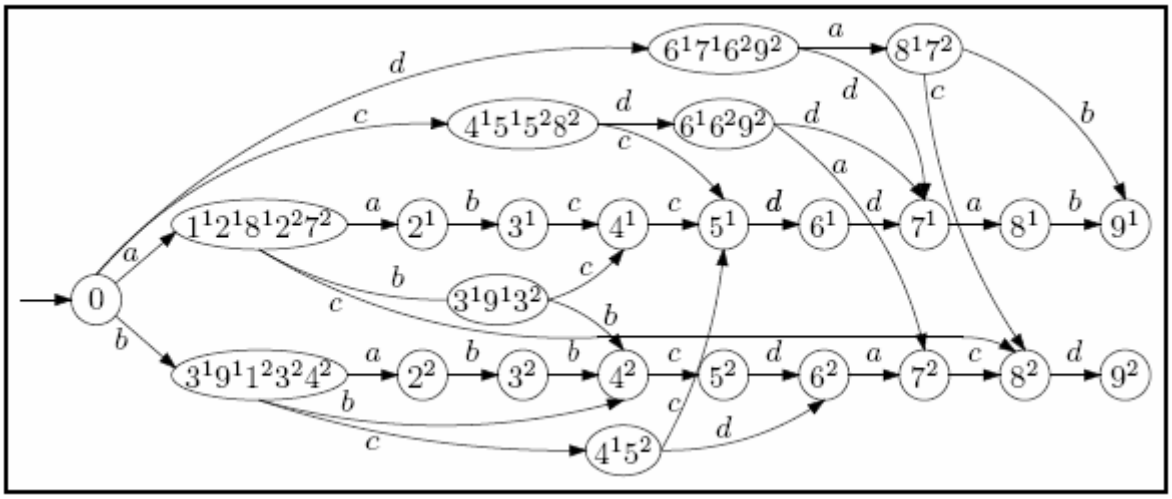

Figure 42
Finite deterministic factor automaton resulting from union of individual automata for each string in a set [AHIM+06]

Specifically, when searching for common patterns between length *i* and *j* amongst a set of strings $S = \{S_1, S_2, \ldots S_n\}$, the authors create an automaton for each string in the set that accepts all factors within the length boundaries created by *i* and *j*. Then each of these automata are merged together (unioned) and, if the result is non-deterministic, converted to be deterministic (see Figure 42). The set of factors accepted by this automaton are then used to create a second set of automata for the strings in S, where each automaton is created using the factors identified in the previously intersected global automaton. The second set of automata are once again merged with the end result being that each repeated factor occurring in all of the strings within S can be identified by the final deterministic union automaton; that is, any string accepted by this last automaton will represent a repeating factor in the set. The authors use this approach to allow for string searching with gaps. Other papers that explore the use of automata for the searching of patterns within strings are given by Crochemore et al. [CCGJ+93], Raffinot [Raf97], Navarro and Raffinot [NR98], and by Hosoya et al. [HSIM05] for use in lyrical recognition.

5.4.10 Correlative Matrices

Correlative matrices, also called self-similarity matrices, allow patterns to be discovered within a string of musical features by comparing that string to itself. These devices are constructed by placing a musical string down both the x and y axes

of a matrix, to allow placement of numerical values that indicate the current and past matching information at various points in the string (see Figure 43).

	C6	Ab5	Ab5	C6	C6	Ab5	Ab5	C6	Db5	C6	Bb5	C6
C6	—			1	1			1		1		1
Ab5		—	1			2	1					
Ab5			—			1	3					
C6				—	1			4		1		1
C6					—			1		1		1
Ab5						—	1					
Ab5							—					
C6								—		1		1
Db5									—			
C6										—		1
Bb5											—	
C6												—

Figure 43
An example correlative matrix from [HCL98]

The resulting matrix can be analyzed for patterns forming in the rows and columns; often these algorithms traverse the comparison space diagonally looking for increasing values and value groupings as an indication of repeating patterns. Correlative matrices can be quite spatially expensive for the induction of patterns involving long musical strings, due to the inherent $O(n^2)$ complexity imposed by the matrix itself. Use of correlative matrices can be found in a series of papers by Chen, Hsu and Liu [HCL98, LHC99, HLC01], as well as in Smith and Medina [SM01].

5.4.11 A Word on Melodic Segmentation

A number of approaches attempt to identify patterns by segmenting a melodic surface into perceptibly-bounded parts in a pre-processing stage for an induction algorithm. This intrinsically increases the relevance of patterns that are checked for repetition, as those that cross segmentation boundaries are omitted. Additionally, musical segments can be processed independently, allowing for complicated induction algorithms to be run incrementally or in parallel. One of the most important tasks in melodic segmentation is determining how to identify appropriate boundaries in music; the use of rests, bar divisions, and in many cases individual note weighting functions can all help to contribute to a boundary strength profile that can suggest good segmentation points. Melodic segmentation itself is an entire MIR research topic in its own right, and is omitted from this paper due to scope constraints. Still, melodic segmentation algorithms and approaches can be found in [Cam03, Cam06, CT04, FN03, Meu02, MS03, SKKK03].

6.0 Summary and Discussion

This literature survey examined a wide range of papers dealing with topics relating to musical information retrieval. The most common methods of representing music were explored, with a particular emphasis on storing the various types of useful musical features. Several prominent approaches and algorithms were outlined, both in the areas of measuring musical similarity, and also the process of extracting information from musical passages expressed in a variety of sampled and non-sampled formats. While this area of MIR is well researched by the academic community, there exists room for additional investigation into possible methods of extracting and storing harmonic chord progressions; it is likely that a lack of useful algorithms in this field is directly related to the difficulty of analyzing chordal progressions based on notational information alone, a significant problem considering the most common file format (MIDI) lacks many of the useful features of fully scored music. In general, MIR researchers would greatly benefit from the emergence of a better standard music format such as those of the digital score family; it is highly unlikely that such a format would emerge as a de facto standard on its own, and thus algorithms that can accurately and fully convert MIDI to more complex digital formats would be invaluable.

Several methods of query capture and representation were also explored, giving insight into some of the common problems associated with attempting to interpret musical input from users with varying skills and abilities. The most important of these paradigms is query by humming, since the human voice is the only universal instrument available to all users of any race and/or ethnic background. Still, query-by-humming is particularly error-prone during the transcription phase of an MIR system, and research has revealed that any successful query mechanism would require significant knowledge into both the common errors made by users, as well as an in-depth understanding of issues in human psychoacoustics.

While it is clear from the topics explored that much work has been done in the field of MIR, some areas remain that have not yet been sufficiently explored. First, more authors should attempt to coordinate the advantages of multiple approaches to musical pattern matching in a single MIR framework. For example, when searching for patterns in music, a smooth contour approach often provides excellent accuracy, though at a high computational cost. One solution to this would be to use the advantages of reduced contours and smooth contours together; queries could first be transcribed into a reduced contour and then searched among a reduced contour database. This would result in a superset of the record matches that a smooth contour approach would give. From this superset, an MIR system could then apply smooth contour analysis for further refinement; thus, the reduced contour approach would act as a filter to decrease the computational burden of applying smooth contour analysis to the entire MIR database. It is conceptually feasible that this approach would increase efficiency while incurring little or no cost in matching accuracy.

This is particularly evident in the unexplored area of binary pitch contours, or those pitch contours that can be represented in an alphabet of size 2 {0, 1}. The closest resemblance to this approach would be the U,D,S approach by Ghias et al. [GLCS95] outlined in Section 2.3.3.1, with the difference being that the S symbol is removed. Consider a contour where a 1 is placed if a pitch interval represents an increase or the same pitch, and a 0 is placed if the pitch interval represents a decrease (see Figure 44).

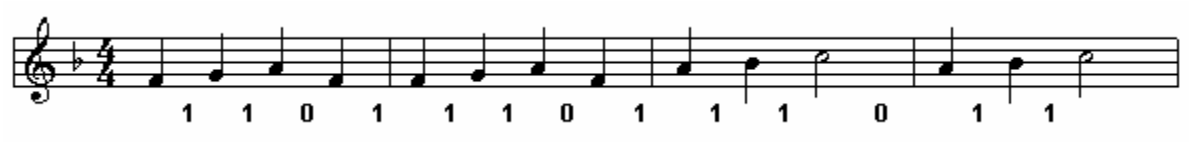

Figure 44
A possible binary pitch contour for Frère Jacques

While this type of contour suffers from massive data loss in the pitch domain, it is particularly well-suited to use the advantages of bit parallelism. This is especially true with the advent of native 64-bit processing hardware, which could handle a musical phrase of up to 64 pitch intervals in a single 64-bit value; studies have shown that the average real-world theme melody for large databases of music hovers between 55 and 70 notes [SMW97, LHC99]. For example, if the contour in Figure 44 is taken to be a 64-bit value with 0's padded to the least significant bits, the decimal equivalent value would be 15985526877351575552. Even the storage of pitch contours in 32-bit integers could serve many purposes, since themes could be broken into parts or stored as a series of melodic subsets. By comparing melodic contours using binary operations, large databases of musical themes could be searched very quickly without requiring any data conversion. As mentioned above, if the resulting matches are not sufficiently precise, refinement of the set with step-leap or smooth contour comparisons could serve as a second stage and even third stage in the matching process. Alternatively, the pairing of reduced rhythmic contours with binary pitch contours could allow very fast searching in both the pitch and rhythmic domain, as an enriched initial step. As stated in Section 2.4.4, no substantial research involving reduced rhythmic contours has been done.

Other potential areas of research includes investigations into new types and uses for musical histograms, as studies exploring these constructs seem to yield positive results in general. The combination of multiple histograms in multiple feature domains (rhythm, pitch, tempo change, etc..) could serve as an efficient and useful method for pattern matching and extraction, or for further investigations into genre classification similar to the work of [TEC02]. Another area open to study is the identification of chordal progressions in music, where notes could be used to harmonically analyze local tonic values and other intervals, perhaps as an aid for melodic segmentation. Some effort in this area has been done by Klapuri [Kla03] (see Section 3.2.4). Finally, limited research has been done to investigate how musical

patterns can be stored and used for automated musical composition; for some papers that cover these issues, see [KF06, Mir01, Mir04, VDF04].

7.0 References

Note: For the reader's convenience, a large (but not exhaustive) list of electronic URLs for these and other documents can be found at: http://www.demopoulos.org/?cat=4

[Ada04] Adams, N. H. (2004) Time Series Representations for Music Information Retrieval, University of Michigan

[ABSW04] Adams, N. H., Bartsch, M., Shifrin J., and Wakefield, G. (2004) Time series alignment for music information retrieval. In *Proceedings of the International Conference on Music Information Retrieval*, 303–310

[AB02] Adams, N. H., & Bolton, R.J. (2002) Eds. Lecture Notes In Computer Science, vol. 2447. Springer-Verlag, London, 190-198

[AMW05] Adams, N. H., Marquez, D., and Wakefield, G. (2005) "Iterative Deepening for Melody Alignment and Retrieval," In *Proc. Int. Conf. on Music Inf. Retrieval (ISMIR)*, London, UK, September

[AHIM+06] Antoniou, P., Holub, J., Iliopoulos, C.S., Melichar, B., Peterlongo, P. (2006) Finding Common Motifs with Gaps using Finite Automata. In O. H. Ibarra, Hsu-Chun Yen (eds.) *Proceedings of the 11th International Conference on Implementation and Application of Automata (CIAA2006)*, National Taiwan University, Taipei, Taiwan, LNCS 4094, Springer-Verlag, pp. 69-77

[Bak97] Bakhmutova, I. V., Gusev, V.D. and Titkova, T.N. (1997) The Search for Adaptations in Song Melodies. *Computer Music Journal*, 12(1):58-67

[BM48] Barlow, H., & Morgenstern, S. (1948) A Dictionary of Musical Themes. Crown Publishers

[BEWS04] Batke, J., Eisenberg, G., Weishaupt, P., Sikora, T. (2004) A Query by Humming system using MPEG-7 Descriptors. In *Proceedings of the 116th AES Convention*, 2004

[BMS00] Bello, J. P., Monti, G., Sandler, M. (2000) Techniques for Automatic Music Transcription. In *International Symposium on Music Information Retrieval*, 2000

[BVW06] Bosma, M. Veltkamp, R. C., Wiering, F. (2006) Muugle: A modular music information retrieval framework. In *ISMIR Proceedings*, 2006.

[BZ91] Brown, J. C., and Zhang, B. (1991) Musical frequency tracking using the methods of conventional and narrowed autocorrelation. *J. Acoust. Soc. Am. 89*, 2346-2354

[Bro80] Brook, B. S. Thematic catalogue. (1980) In *The new Grove dictionary of music and musicians*, ed. Stanley Sadie. London: Macmillan Publishers, 1980.

[BM77] Boyer, R. S., & Moore, J.S. (1977) A fast string searching algorithm. *Commun. ACM 20*, 10, 762-772

[Cam01] Cambouropoulos, E. (2001) Automatic pitch spelling: From numbers to sharps and flats. In *VIII Brazilian Symposium on Computer Music (SBC&M 2001)*, Fortaleza, Brazil

[Cam00a] Cambouropoulos, E. (2000) Extracting significant patterns from musical strings: some interesting problems. In *Proceedings of the London String Days workshop*, King's College London and City University

[Cam00b] Cambouropoulos, E. (2000) From MIDI to Traditional Musical Notation. In *Proceedings of the AAAI Workshop on Artificial Intelligence and Music: Towards Formal Models for Composition, Performance and Analysis 30 July - 3 Aug 2000*, Austin, Texas

[Cam06] Cambouropoulos, E. (2006) Musical Parallelism and Melodic Segmentation: A Computational Approach. *Music Perception 23(3)*:249-269

[Cam03] Cambouropoulos, E. (2003) Musical pattern extraction for melodic segmentation. In *Proceedings of the ESCOM Conference 2003*, Hannover, Germany

[CCI01] Cambouropoulos, E., Crawford, T., Iliopoulos, C. S. (2001) Pattern processing in melodic sequences: challenges, caveats and prospects. In *Computers and the Humanities 35 (1)*, pp. 9-21

[CCIM+05] Cambouropoulos, E., Crochemore, M., Iliopoulos, C. S., Mohamed, M., and Sagot, M. (2005) A Pattern Extraction Algorithm for Abstract Melodic Representations that Allow Partial Overlapping of Intervallic Categories. In, T. Crawford, M. Sandler, editors, *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005)*, pp. 167–174

[CCIM+99] Cambouropoulos, E., Crochemore, M., Iliopoulos, C. S., Mouchard, L., Pinzon, Y. J. (1999) Algorithms for computing approximate repetitions in musical sequences. In R. Raman and J. Simpson, editors, *Proceedings of the 10th Australasian Workshop on Combinatorial Algorithms*, pages 129-144, Perth, WA, Australia

[CT04] Cambouropoulos E., & Tsougras, C. (2004) Influence of Musical Similarity on Melodic Segmentation: Representations and Algorithms. In *Proceedings of the International Conference on Sound and Music Computing (SMC04)*, Paris, France

[CC98] Chen, J. C. C., Chen, A. L. P. (1998) Query by Rhythm: An Approach for Song Retrieval in Music Databases. *RIDE 1998*: 139-146

[CEMS00] Clausen, M., Engelbrecht, R., Meyer, D., and Schmitz, J. (2000) Proms: A web-based tool for searching in polyphonic music. In *Proceedings of the 1st International Symposium on Music Information Retrieval (ISMIR 2000)*

[CKE01] Clausen, M., Kurth, F., Engelbrecht, R. (2001): Context-based Retrieval in MIDI and Audio. In Fellner, D.W., Fuhr, N., Witten, I. (Eds.): *ECDL Workshop: Generalized Documents*. Darmstadt, September

[Col04] Collins, N. (2004) Beat Induction and Rhythm Analysis for Live Audio Processing: 1st Year PhD Report. Centre for Science and Music, Faculty of Music, University of Cambridge. Released 18/06/2004

[CIR98] Crawford, T., Iliopoulos, C. S., and Raman, R. (1998) String Matching Techniques for Musical Similarity and Melodic Recognition. *Computing in Musicology 11*, 73-100

[Cro81] Crochemore, M. (1981) An Optimal Algorithm for Computing the Repetitions in a Word. *Information Processing Letters 12*, 244-250

[CCGJ+93] Crochemore, M., Czuma, A., Gasieniec, L., Jarominek, S., Lecroq, T., Plandowski, W., Rytter, W. (1993) Fast practical multi-pattern matching. *Rapport 93-3*, Institut Gaspart Monge, Université de Marne la Vallée

[CILP+02] Crochemore, M., Iliopoulos, C. S., Lecroq, T., Pinzon, Y. J., Plandowski, W., and Rytter, W. (2003) Occurrence and substring heuristics for δ-matching. *Fundam. Inf. 56*, 1,2, 1-21, July

[CIMR+02] Crochemore, M., Iliopoulos, C., Makris, C., Rytter, W., Tsakalidis, A., Tsichlas, K. (2002) Approximate string matching with gaps. *Nordic Journal of Computing 9(1)*:54-65, Spring

[CINP03] Crochemore, M., Iliopoulos, C., Navarro, G., Pinzon, Y. (2003) A bit-parallel suffix automaton approach for (δ, γ)-matching in music retrieval. In *Proc. 10th International Symposium on String Processing and Information Retrieval (SPIRE'03)*, Springer-Verlag, pp. 211-223

[DBPH+07] Dannenberg, R. B., Birmingham, W. P., Pardo, B., Hu, N., Meek, C., Tzanetakis, G. (To appear February 2007) A Comparative Evaluation of Search Techniques for Query-by-Humming Using the MUSART Testbed," *Journal of the American Society for Information Science and Technology, 58(3)*

[DO06] De Poli, G., & Orio, N. (2006) Music Information Processing. Algorithms for Sound and Music Computing, Chapter 6: Publication Status: UNKNOWN

[Dix00] Dixon, S., & Cambouropoulos E. (2000) Beat Tracking with Musical Knowledge. Accepted for *ECAI 2000*, Berlin

[DR01] Doraisamy, S., & Rüger, S. M. (2001) An approach toward a polyphonic music retrieval system. In J. S. Downie and D. Bainbridge, editors, *Proceedings of the 2nd Annual International Symposium on Music Information Retrieval (ISMIR)*, pages 187-193, Indiana University, Bloomington, Indiana, October

[Dow78] Dowling, W. J. (1978) Scale and contour: Two components of a theory of memory for melodies. *Psychological Review, 85(4)*:341-354

[DN00] Downie, S., Nelson, M. (2000) Evaluation of a Simple and Effective Music Information Retrieval Method. In *Proc. of the ACM-SIGIR*, Athens, GR, 73-80

[FN03] Ferrand, M., & Nelson, P. (2003) Unsupervised learning of melodic segmentation: A memory-based approach. In *Proceedings of the 5th Triennial ESCOM Conference*, Hanover, Germany, pp. 141-144

[Foo99] Foote, J. (1999) An Overview of Audio Information Retrieval. *Multimedia Systems, 7(1)*, 307–328

[GKM03] Gómez, E., Klapuri, A., Meudic, B. (2003) Melody Description and Extraction in the Context of Music Content Processing. *Journal of New Music Research Vol.32.1*, 2003.

[GSOP+06] Gómez, E., Streich, S., Ong, B., Paiva, R., Tappert, S., Batke, J., Poliner, G., Ellis, D., Bello, J. (2006) A Quantitative Comparison of Different Approaches for Melody Extraction from Polyphonic Audio Recordings. *MTG-TR-2006-01*

[GLCS95] Ghias, A., Logan, J., Chamberlin, D., Smith, B. C. (1995) Query by humming: Musical information retrieval in an audio database. In *Proc. of ACM Multimedia*, 231-236

[Hof01] Hofmann-Engl, L. (2001) Towards a cognitive model of melodic similarity. In *Proceedings of the 2nd Annual International Symposium on Music Information Retrieval*, Bloomington, Indiana, pp 143–151

[HHRK98] Hoos, H., Hamel, K., Renz, K., Kilian J. (1998) The GUIDO Notation Format -- A Novel Approach for Adequately Representing Score-Level Music. In *Proceedings ICMC 98*. 451–454

[HSIM05] Hosoya, T., Suzuki, M., Ito, A., Makino, S. (2005) Lyrics recognition from a singing voice based on finite state automaton for music information retrieval. In *Proc. ISMIR 2005*, pp. 532–535

[HCL98] Hsu, J., Chen, A. L. P, and Liu, C. (1998) Efficient repeating pattern finding in music databases. In *Proceedings of the Seventh international Conference on information and Knowledge Management CIKM '98*, Bethesda, Maryland, United States, November 02-07. ACM Press, New York, NY, 281-288

[HLC01] Hsu, J., Liu, C., Chen, A. L. P. (2001) Discovering nontrivial repeating patterns in music data. *IEEE Transactions on Multimedia, vol.3, no.3*, pp.311-325, Sept

[HD02] Hu, N., & Dannenberg, R. B. (2002) A Comparison of Melodic Database Retrieval Techniques Using Sung Queries. In *Joint Conference on Digital Libraries, Association for Computing Machinery*

[HDL02] Hu, N., Dannenberg, R. B., Lewis, A. L. (2002) A Probabilistic Model of Melodic Similarity. In *Proceedings of the International Computer Music Conference (ICMC'02)*, Gotheborg, Sweden, September

[ILMP00] Iliopoulos, C., Lecroq, T., Mouchard, L., Pinzon, Y. (2000) Computing approximate repetitions in musical sequences. In Proc. of *Prague Stringology Club Workshop (PSCW'00)*, pages 49–59

[KBT04] Kapur, A., Benning, M., Tzanetakis, G. (2004) Query-By-Beat-Boxing: Music Retrieval for the DJ. In *Proc. of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, Barcelona, October 10-14

[KNKT95] Kashino, K., Nakadai, K., Kinoshita, T., and Tanaka, H. (1995) Organization of Hierarchical Perceptual Sounds: Music Scene Analysis with Autonomous Processing Modules and a Quantitative Information Integration Mechanism.
In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 1, 158–164

[KF06] Khalifa, Y. M. A., Foster, R. (2006) A Two-Stage Autonomous Evolutionary Music Composer. *EvoWorkshops 2006*: 717-721

[Kla03] Klapuri, A. P. (2003) Musical meter estimation and music transcription. In *Proc. Cambridge Music Processing Colloquium*, pages 40-5

[Kla00] Klapuri, A. P. (2000) Qualitative and quantitative aspects in the design of periodicity estimation algorithms. In *Proc. European Signal Processing Conference*, Tampere, Finland, September

[KMP77] Knuth, D. E., Morris, J. H., Pratt, V. R. (1977) Fast pattern matching in strings. *SIAM Journal on Computing 6*, 1, 323-350

[KY01] Koh, J., & Yu, W. D. (2001) Efficient Feature Mining in Music Objects. In *Proceedings of the 12th international Conference on Database and Expert Systems Applications (September 03 - 05, 2001)*. Mayr, H. C., Lazanský, J., Quirchmayr, G., Vogel, P. Eds. Springer-Verlag, London, 221-231

[Kru90] Krumhansl, C. L. (1990) Cognitive Foundations of Musical Pitch, Oxford University Press, New York

[KS79] Krumhansl, C. L., & Shepard, R. N. (1979) Quantification of the hierarchy of tonal functions within a diatonic context. *Journal of Experimental Psychology: Human Perception and Performance, 5(4)*: 579-594

[LK94] Large, E. W., & Kolen, J. F. (1994) Resonance and the perception of musical meter. *Connection Science, 6*, 177 – 208

[Lar05] Lartillot, O. (2005) Multi-Dimensional Motivic Pattern Extraction Founded on Adaptive Redundancy Filtering. In *Journal of New Music Research Vol. 34*, No. 4, pp. 375 – 393

[Lar03] Lartillot, O. (2003) Perception-based advanced description of abstract musical content. In Izquierdo, E., ed., *Digital Media Processing for Multimedia Interactive Services*

[LL98] Lemström, K., & Laine, P. (1998) Musical Information Retrieval Using Musical Parameters. In *Proc. 1998 International Computer Music Conference (ICMC'98)*, pp. 341-348

[LJ83] Lerdahl, F., Jackendoff, R. A. (1983) Generative Theory of Tonal Music. *MIT Press*, Cambridge, MA

[Lev66] Levenshtein, V.I. (1966) Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady, vol. 10*, pp. 707-710

[Lin67] Lincoln, H. B. (1967) Some criteria and techniques for developing computerized thematic indices. In Heckmann, editor, *Electronische Datenverarbeitung in der Musikwissenschaft*, Regensburg

[LHC99] Liu, C., Hsu, J., Chen, A. L. P. (2005) Efficient theme and non-trivial repeating pattern discovering in databases. In *Proc. of ICDE 99*, Sydney, Australia, pp. 14-21

[LWC05] Liu, N. H., Wu, Y. H., and Chen, A. L. P. (2005) An Efficient Approach to Extracting Approximate Repeating Patterns in Music Databases. In *Proceedings of International Conference on Database Systems for Advanced Applications*

[Lon87] Longuet-Higgins, H. C. (1987) The perception of melodies. In H. Christopher Longuet-Higgins, editor, *Mental Processes: Studies in Cognitive Science*, pages 105–129. British Psychological Society/MIT Press, London, England and Cambridge, Mass

[MD01] Mazzoni, D., & Dannenberg, R. B. (2001) Melody matching directly from audio. In *2nd Annual International Symposium on Music Information Retrieval*, Bloomington, Indiana, USA

[MM04] McKinney, M., & Moelants, D. (2004) Extracting the perceptual tempo from music. In *Proceedings of the 5th International Symposium on Musical Information Retrieval 2004*. Barcelona: Universitat Pompeu Fabru,

[MB02] Meek, C., & Birmingham, W. P. (2002) Johnny can't sing: A comprehensive error model for sung music queries. In *Proceedings of the Third International Conference in Music Informatics Retrieval*

[MB01] Meek, C. and Birmingham, W. P. (2001) Thematic Extractor. In *2nd Annual International Symposium on Music Information Retrieval*, Bloomington, Indiana University, Indiana, 119-128

[Mer04] Meredith, D. (2004) Comparing Pitch Spelling Algorithms on a Large Corpus of Tonal Music. In U.K. Wiils (ed). *Computer Music Modeling and Retrieval*. Esbjerg, Denmark: Springer-Verlag LNCS #3310

[MLW02] Meredith, D., Lemstrom, K., Wiggins, G. A. (2002) Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research, 31(4)*:321--345

[Meu03] Meudic, B. (2003) Musical pattern extraction: from repetition to musical structure. In *Proceedings of CMMR (Computer Music Modeling and Retrieval)*, Montpellier, May

[Meu02] Meudic, B. (2002) A causal algorithm for beat tracking. *2nd conference on understanding and creating music*, Caserta, Italy

[MS03] Meudic, B., St James, E. (2003) Automatic Extraction of Approximate Repetitions in Polyphonic Midi Files Based on Perceptive Criteria. *Computer Music Modelling and Retrieval*, LNCS 2771, Springer-Verlag

[Mir04] Miranda, E.R. (2004) At the Crossroads of Evolutionary Computation and Music: Self-Programming Synthesizers, Swarm Orchestra and Origins of Melody. *Evolutionary Computation, Vol. 12*, No. 2, pp. 137-158

[Mir01] Miranda, E.R. (2001) Composing Music Using Computers. Focal Press, Oxford, UK

[MS90] Mongeau, M., & Sankoff, D. (1990) Comparison of Musical Sequences. In Hewlett, W., & Selfridge-Field, E. eds. *Melodic Similarity Concepts, Procedures, and Applications*, MIT Press, Cambridge

[NR98] Navarro, G., & Raffinot, M. (1998) Fast and flexible string matching by combining bit-parallelism and suffix automata. Technical Report TR/DCC-98-4, Dept. of Computer Science, Univ. of Chile

[NO04] Neve, G., Orio, N. (2004) Indexing and Retrieval of Music Documents through Pattern Analysis and Data Fusion Techniques. In *Proc. of the International Conference on Music Information Retrieval*, Barcelona, ES, pp. 216-223

[Par05] Parker, C. L. (2005) Applications of Binary Classification and Adaptive Boosting to the Query-By-Humming Problem. *ISMIR 2005*: 245-251

[Par94] Parncutt, R. (1994) A Perceptual Model of Pulse Salience and Metrical Accent in Musical Rhythms. *Music Perception 11*, 409-464

[PK02] Paulus, J., & Klapuri, A. (2002) Measuring the similarity of rhythmic patterns. In *Proceedings of the International Conference on Music Information Retrieval*

[PG77] Piszczalski, M., & Galler, B. F. (1977) Automatic Music Transcription. *Computer Music Journal,* 1, 24-31

[PE85] Povel, D. J., & Essens, P. J. (1985) Perception of temporal patterns. *Music Perception*, 2, 411-441

[Raf97] Raffinot, M. (1977) On the multi-backward dawg matching algorithm (MultiBDM). In *Proceedings WSP'97*, Carleton University Press, pp. 149-165

[Ren02] Renz, K. (2002) Algorithms and data structures for a music notation system based on GUIDO music. PhD thesis, Universität Darmstadt

[Rol98] Rolland, P. Y. (1998) FlExPat: A Novel Algorithm for Musical Pattern Discovery. In *Proceedings of the XII Colloquium in Musical Informatics*, Gorizia, Italy

[Rol01a] Rolland, P. Y. (2001) FlExPat: Flexible Extraction of Sequential Patterns. In *Proceedings of the 2001 IEEE international Conference on Data Mining (November 29 - December 02, 2001)*. N. Cercone, T. Y. Lin, and X. Wu, Eds. ICDM. IEEE Computer Society, Washington, DC, 481-488

[Rol01b] Rolland, P. Y. (2001) Introduction: Pattern Processing in Music Analysis and Creation. *Computers and the Humanities, Volume 35, Number 1*, February 2001, pp. 3-8(6)

[Rol00] Rolland, P. Y. & Ganascia J. G. (2000) Musical Pattern Extraction and Similarity Assessment. In E. M. Miranda (Ed.), *Readings in Music and Artificial Intelligence*, Harwood Academic 115-144

[RG02] Rolland, P. Y. & Ganascia, J. G. (2002) Pattern Detection and Discovery: The Case of Music Data Mining. In *Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery (September 16 - 19, 2002),* D. J. Hand, N. M.

[RRG99] Rolland, P. Y., Raškinis, G., Ganascia, J. G. (1999) Musical content-based retrieval: an overview of the Melodiscov approach and system. In *Proceedings of the Seventh ACM international Conference on Multimedia (Part 1)*, Orlando, Florida, United States, October 30 - November 05. MULTIMEDIA '99. ACM Press, New York, NY, 81-84

[Ros92] Rosenthal, D. (1992) Emulation of Human Rhythm Perception. *Computer Music Journal 16(10)*:64-76

[Row93] Rowe, R. (1993) Interactive Music Systems (Machine Listening and Composing). Cambridge, Mass: The MIT Press

[Sch98] Scheirer, E. (1998) Tempo and Beat Analysis of Acoustic Musical Signals. *Journal of the Acoustical Society of America, 103(1)*:588,601, January

[Sep01] Seppänen, J. (2001) Computational models of musical meter recognition. Master's thesis, Dept. Information Technology, Tampere University of Technology, August

[SKKK03] Shin, C., Ku, K., Kim, K., Kim, Y. (2003) Automatic Construction of Theme Melody Index from Music Database for Fast Content-Based Retrievals. *ECIR 2003*: 605-612

[SW81] Smith, T. F., & Waterman, M. S. (1981) Identification of common molecular subsequences. *Journal of Molecular Biology 147*, 195-197

[SMW97] Smith, L. A., McNab, R. J., Witten, I. H. (1997) Music Information Retrieval Using Audio Input. In *Proceedings of the AAAI: Intelligent Integration and Use of Test, Image, Video and Audio Corpora*, Stanford, CA, pp. 12-16

[SM01] Smith, L., & Medina. R. (2001) Discovering themes by exact pattern matching. In *Proc. of International Symposium on Music Information Retrieval*, pp 31-32

[TY99] Takasu, A., Yanase, T., Kanazawa, T., and Adachi, J. (1999) Music Structure Analysis and Its Application to Theme Phrase Extraction. In *Proceedings of the Third European Conference on Research and Advanced Technology For Digital Libraries (September 22 - 24, 1999)*. Abiteboul, S., & Vercoustre, A. Eds. Lecture Notes In Computer Science, vol. 1696. Springer-Verlag, London, 92-105

[Tem97] Temperley, D. (1997) An Algorithm for Harmonic Analysis. *Music Perception, 15(1)*:31-68

[Tem01] Temperley, D. (2001) The Cognition of Basic Musical Structures. MIT Press, Camb, MA

[TB02] Temperley, D., & Bartlette, C. (2002) Parallelism as a Factor in Metrical Structure. *Music Perception, 20(2)*:117-149

[TEC02] Tzanetakis, G., Ermolinskiy, A., Cook, P. (2002) Pitch Histograms in Audio and Symbolic Music Information Retrieval. In *Proc. 3rd ISMIR*: 31-38

[UZ98] Uitdenbogerd, A. L., & Zobel, J. (1998) Manipulation of Music for Melody Matching. *In Proc. ACM Multimedia '98*, pp. 235-240

[VDF04] Verbeurgt, K., Dinolfo, M., and Fayer, M. (2004) Extracting patterns in music for composition via Markov chains. In *Proceedings of the 17th international Conference on innovations in Applied Artificial intelligence (Ottawa, Canada, May 17 - 20, 2004)*. Orchard, R., Yang, C., Ali, M. Eds. Lecture Notes In Computer Science. Springer Verlag, 1123-1132

[WR03] Wieczorkowska, A. A., & Ras, Z. W. (2003) Editorial—Music Information Retrieval. *Journal of Intelligent Information Systems 21*, 5-8, July