A MUSIC DATABASE AND QUERY SYSTEM FOR RECOMBINANT COMPOSITION

James B. Maxwell
School for the Contemporary Arts
Simon Fraser University, Burnaby, B.C.
jbmaxwel@sfu.ca

Arne Eigenfeldt
School for the Contemporary Arts
Simon Fraser University, Burnaby, B.C.
arne_e@sfu.ca

ABSTRACT

We propose a design and implementation for a music information database and query system, the MusicDB, which can be used for Music Information Retrieval (MIR). The MusicDB is implemented as a Java package, and is loaded in MaxMSP using the mxj external. The MusicDB contains a music analysis module, capable of extracting musical information from standard MIDI files, and a search engine. The search engine accepts queries in the form of a simple six-part syntax, and can return a variety of different types of musical information, drawing on the encoded knowledge of musical form stored in the database.

1. INTRODUCTION

Inspired by the analysis techniques developed by David Cope for his work in the field of algorithmic composition by "music recombinance", the MusicDB [11] was originally designed as an analysis and data-retrieval back-end for an interactive music composition system. One of the primary motivations behind the design of this system was to allow the user to create original musical works, drawing on the musical language exemplified by their existing works, but maintaining as much flexibility as possible. We were drawn to Cope's notion of music recombinance because of its proven capacity to replicate musical style, which we felt would allow the user to easily incorporate the software into their existing compositional practice, and provide the greatest continuity between new works composed with the system and existing works. However, since the principle of paraphrasing which underlies the notion of "strict" recombinance can be somewhat of a limitation on originality, we felt it was important to develop a system that could promote musical continuity without relying exclusively on verbatim quotations of musical fragments for its base materials.

As the music recombinance approach to composition is clearly "data-driven" [6], the fundamental design of the MusicDB is well suited for use in the field of MIR. The software was designed to parse, analyse, and store information from symbolic musical representations—specifically, from standard MIDI files of 'scored' musical works—in a way which acknowledges similarities across multiple source works, but which also retains a formal 'map' of the original structure of each individual work. Single MIDI files of works are analysed by the software, and the analysis output is saved in the form of proprietary ".speac" data files. These files can then be loaded into a "session", consisting of an arbitrary number of analysed works. The database built during a session represents the compilation all of the analysed musical material into a single data structure, making it possible for the system to reveal inter-relationships between the elements of a potentially large body of analysed works.

1.1. Organization by Hierarchy

The MusicDB uses Cope's "SPEAC analysis" system to build a hierarchical analysis of each source work. A more detailed description of SPEAC is given later in the paper, and can also be found in numerous publications [3, 4, 5, 6]. This hierarchical analysis builds a tree-like structure for the representation of the musical form, in which the formal development of the music is segmented according to continuous changes observed over a variety of analysis parameters. In the MusicDB, this type of analysis is applied to each individual work, and is also used as an organizational system for the final session database. The end result is a data structure that associates the content of all the analysed works according to their overall pattern of formal development, regardless of their unique surface details. That is, the introductory, developmental, and concluding formal sections of all the stored works will be grouped together, regardless of the specific style or characteristics of each individual work. This allows the user to look at the formal contour of works, and find formal correlations between works, regardless of superficial and/or perceptual differences in musical content.

1.2. MusicDB Objectives

The MusicDB differs from Cope's work in that we have sought to develop a *component* for data analysis, storage, and recall, to be used as a tool by algorithmic composition

system designers. In particular, we have designed the MusicDB to accept input searches, and provide output data, at varying degrees of musical abstraction. While the system *can* provide complete, verbatim quotations of musical fragments from the analysed source works, it is not limited to such quotations. It can also be queried for more abstract representations, like melodic contour [12], chroma [13], kinesis [6], periodicity [10], and so on, thus offering the possibility of using musical knowledge provided by example works as a formal guideline for composition systems which are not otherwise based in music recombinance. In this sense, the MusicDB could be used strictly as a music analysis tool, or as a parameter control module for an agent-based [1, 7, 8], or fuzzy logic-based, system [2]—i.e., if such a system took kinesis (general activity level) and harmonic tension as parameters, such parameters could be provided by the MusicDB in a way which modelled the formal development exhibited by the analysed works.

2. IMPLEMENTATION

2.1. Music Analysis

The MusicDB breaks music apart using a hierarchical structuring of objects, which describe the score both vertically and horizontally. Beginning at the lowest level in the horizontal hierarchy, there are Events, Chords, and VoiceSegments. The vertical view of the score is described primarily by Group objects (though Chords obviously imply a vertical aspect as well), and at the top of the hierarchy both horizontal and vertical elements of the score are combined into a composite object called a Phrase (see Figure 1).

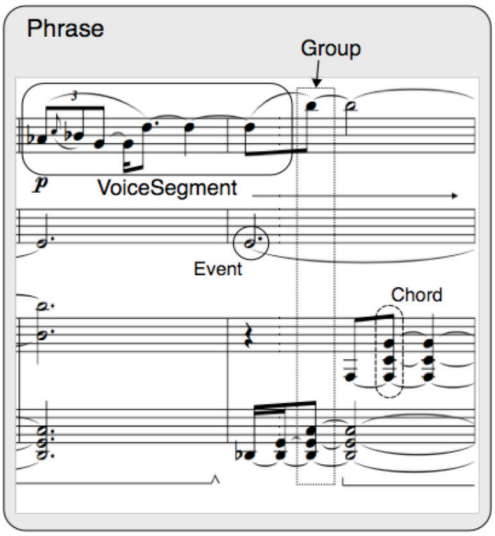

Figure 1. Music descriptors used by the MusicDB

An *Event* is a singular musical sound, occurring at a specific point in time[1]. *Chords* are two or more events, which occur at the same onset time (ED)[2], and a *VoiceSegment* is a horizontal sequence of Events and/or Chords, confined to a single MIDI channel. The decision to keep the voices in polyphonic parts together, where such parts are confined to a single MIDI channel, was a consequence of our desire to preserve, wherever possible, any special characteristics of idiomatic instrumental writing present in the source music. In the current implementation, VoiceSegments are confined to durations of 3 to 8 Chords or Events.

Groups constitute vertical 'slices' of the score, marked by static harmonic structures. In the MusicDB, the smallest Group duration is 1/8th-note. If the harmony remains static, Groups will extend for the entire duration of that harmony, and will always be lengthened in 1/8th-note increments. Groups are identified by an *entryPitch vector*, which is a sorted set of all pitches in the Group's harmonic structure.

Phrases are the largest component objects in the MusicDB, and are generally made from a combination of VoiceSegments and Groups. As the name suggests, a Phrase is a complete, semantically integrated, segment of music. Phrases are determined as sequences of Groups[3], and also hold references to all VoiceSegments found to intersect their Group sequence.

Cope's technique of SPEAC analysis [6] provides a segmentation of symbolic, or 'scored' music[4], based on the relative tensions of Groups. Groups are given a tension score derived from a set of analysis parameters, and are then assigned SPEAC 'labels' (**S**tatement, **P**reparation, **E**xtension, **A**ntecedent, **C**onsequent) based on their relative scores. A new segment, or Phrase, is created at each **A** to **C** label transition. The same process is then applied to the segmentation of the entire sequence of "foreground" Phrases, thus deriving a sequence of "background" Phrases (see Figure 4) and so on. The resulting SPEAC hierarchy provides a clear formal topography for the analysed work, giving the MusicDB access to the musical materials therein at the level of individual Groups, Phrases (and their component VoiceSegments), or sequences of Phrases (i.e., 'sections'), and also as a complete formal 'tree.' The hierarchical nature of SPEAC analysis is exploited by the MusicDB's search engine, which will be discussed later.

[1] The event's time is taken directly from the MIDI file, and is defined as the event's location in MIDI ticks (i.e., "timestamp").

[2] We use the concept of Entry Delay (ED), to identify the onset time of an event, calculated as the time passed *since* the previous event's onset.

[3] Phrases can also be made from sequences of other Phrases, as will be shown later in the paper.

[4] The SPEAC system is well-suited to the analysis of symbolic representations of music, not to audio recordings.

The MusicDB performs horizontal segmentation of Groups into Phrases (and Phrases into background Phrases) following the above SPEAC method, however, unlike in Cope, the MusicDB also uses a SPEAC-like system for segmenting sequences of events and/or chords on a single MIDI channel to form VoiceSegments. For this purpose, in addition to Cope's analysis parameters of rhythmic tension, duration tension, and "approach tension" (melodic interval tension), we've added measures for dynamic stress and pitch symmetry. Dynamic stress is simply a scaling of the current MIDI velocity value against the running mean velocity, in order to draw attention to strongly accented events. Pitch symmetry is a scaled interval measurement, which returns the absolute value of the current interval divided by 12. The intention is to show the "leapiness" of the melodic line, where low values indicate step-wise movement or repetitions, and values closer to 1.0 indicate leaps (intervals greater than an octave are set to 1.0).

2.2. Data Structure

The three primary objects used in the MusicDB's data structure—VoiceSegments, Groups, and Phrases—are tightly linked by object referencing. VoiceSegments hold a reference to the Phrase in which they originally appeared (their "root" Phrase), and also hold references to the sequence of Groups found to occur over their musical duration. Further, each VoiceSegment stores references to its "preceding" and "target" (following) VoiceSegment. In a similar manner, Groups hold a reference to their root Phrase, references to preceding and target Groups, and a list of VoiceSegments found to be playing during the vertical 'slice' from which the Group was taken. Phrases store references to all VoiceSegments active during their total duration, and the sequence of either Groups (in the case of "foreground" Phrases), or Phrases (in the case of "background" Phrases) from which they are made. Phrases also store references to their parent Phrases, where appropriate. This proliferation of references makes it computationally trivial to locate and extract a great deal of information about the musical form of a piece, given even the smallest aspect of its content. For example, it would be possible, using an iterative search process, to extract an entire instrumental part from a single musical work in the database, given only a short sequence of pitches, rhythmic values (EDs), chords, etc., as input. It is this interconnectivity by object referencing that is exploited by the MusicDB during the search process. As a convenience, all of the above objects also store the name of the source work in which they originally appeared, and VoiceSegments additionally store the instrument name assigned to the MIDI channel from which the VoiceSegment was derived.[1]

We would like to draw attention to the manner in which pitch-related searches are carried out in the MusicDB. For search processes involving specific pitch content—Pitch Lists and Chords—we have adopted Cope's scale of harmonic tension, rather than using Euclidean distance, or some other linear distance metric, as a measure for what we call "pitch distance." Cope assigns float values to each of the intervals within one octave, as follows [6]:

Interval	Value
0	0.0
1	1.0
2	0.8
3	0.225
4	0.2
5	0.55
6	0.65
7	0.1
8	0.275
9	0.25
10	0.7
11	0.9

All intervals greater than one octave are constrained to pitch-classes before their tension value is calculated. This sequence of values is based on the relative consonance of intervals within the octave, roughly following the harmonic series. It should not be viewed as a set of strictly 'tonal' relationships, though tonality generally follows a similar pattern, but rather as a measure of the degree to which a new note could be substituted for a given note, in an existing musical setting. It should be clear, therefore, that this is a metric chosen to apply to a broad range of *compositional* situations, but which would not be appropriate for most serial procedures, or for set theoretic analysis. Our application of the above harmonic tension scale as a pitch distance metric has the positive effect of relating searched items by a more acoustically founded notion of distance[2] than would be reflected by a linear distance measure. For example, using the above scale to find a match for the Pitch List {60, 62, 67} would return the list {67, 69, 60} as a "closer" match than the list {61, 64, 66}, in spite of the fact that the numeric values are clearly more distant, and the melodic contour is not the same. Assuming that this searched Pitch List was extracted from a supporting harmonic context, such a result would likely offer a more suitable alternative than would be provided by a linear distance metric. A search

[1] The instrument name is taken from the "name" metadata item in the MIDI file.

[2] Again, we would like to stress the fact that we sought to find a solution for a wide number of cases, and we are aware that serial and set theoretic compositional approaches will benefit less from the use of this distance metric.

for Pitch Contour, on the other hand, would use Euclidean distance to compare the Pitch Lists, and would thus have the opposite result.

2.3. Query System

Data is extracted from the MusicDB using a six-part query syntax (see Figure 2).

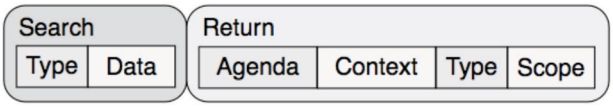

Figure 2. Format of a search query

Searches are carried out on VoiceSegment, Group, and Phrase objects, which hold the following fields[1]:

- **VoiceSegment**: Pitch List, Melodic Interval List, Pitch Contour, Chroma, ED List, ED Contour, Kinesis, Periodicity
- **Group**: Harmonic Interval List, Harmonic Tension, Harmonic Motive
- **Phrase**: Harmonic Tension, Kinesis, Periodicity, Chroma, Pitch Grid

The search **Type** can be chosen from a number of options:

1) PitchList: an ordered list of pitch values
2) MelodicIntervalList: an ordered list of interval values (signed integers)
3) PitchContour: a series of indices giving the relative "height" of each unique pitch in a Pitch List
4) Chroma: a 12-member vector indicating the occurrence-rate of pitch classes
5) EDList: an ordered list of ED values
6) EDContour: contour applied to an ED List
7) Kinesis: the general activity level (0. to 1.)
8) Periodicity: the rhythmic regularity of an ED list (0. to 1.)
9) Chord: an ascending list of pitches
10) HarmonicIntervalList: the intervals between adjacent pitches in a chord (i.e., a major triad is [0, 4, 3])
11) HarmonicTension: the interval tension of a given Chord[2] (0. to 1.)

The search **Data** is a vector of float values, used to represent the search **Type**.

The return **Agenda** (Figure 3) indicates the formal 'motivation' for the search. There are three options: Preceding, Current, and Following. A setting of Current will cause the search to return the requested field (indicated by the return Type) from the object containing the match, while Preceding and Following will return the same field from the preceding or following object, respectively. Beyond returning the matched field itself, the flexibility of the Agenda system makes it possible to extrapolate formal 'causes' and 'effects' relating to the searched data.

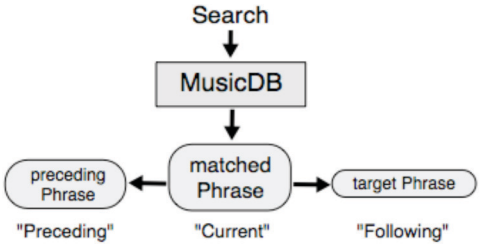

Figure 3. Search Agenda

The return **Context** indicates the immediate musical context of the object holding the requested field. There are three options: Element, Accompaniment, and Setting. A return Context of Element will return only the requested field from the matched object. The Setting option will attempt to provide data for a complete Phrase (i.e., a musical "setting") and will thus return the requested field from all objects referenced by the *root Phrase* holding the matched object[3]. If the requested field is held by a Group object, the *Group sequence* of that object's root Phrase will be returned[4]. For fields held by Groups the Accompaniment option is handled identically to Setting, but for fields held by VoiceSegments, Accompaniment removes the VoiceSegment in which the match was found, returning only the fields from the 'accompanying' VoiceSegments.

Return **Types** include all of the Search Types, with the addition of three more: PitchGrid, HarmonicMotive, and StatisticalRepresentation. A PitchGrid is a 128-member vector indicating the rate of occurrence for each available pitch. A HarmonicMotive is a sequence of Chords (these correspond to the *entryPitch vectors* used to identify Groups[5]), and a StatisticalRepresentation is a vector of statistical analysis values, used internally by the database to measure the similarity between instances of a given class (VoiceSegment, Group, or Phrase).

The final argument, **Scope** (Figures 4 and 5), controls the way in which searches are constrained by the formal structure of the database. A setting of Static (Figure 4) will

[1] All objects also hold a field for their Statistical Representation.

[2] The precise weighting of interval values used to calculate Harmonic Tension is taken from Cope [9].

[3] This is the case whenever the requested field belongs to a VoiceSegment or Group. If it belongs to a Phrase, the Phrase itself provides the return.

[4] The result is equivalent to a "Harmonic Motive".

[5] This is different from the Chord *object*, used by the VoiceSegment class for storing simultaneous pitch events. The term "chord" is used here to fit easily into musical parlance.

search the entire database, beginning at the top of the SPEAC hierarchy, and working its way through the tree. A setting of Progressive (Figure 5) will limit the scope of the search to a particular branch of the tree[1], and thus to a certain 'section' of the encoded musical form. With each completed Progressive search, the focus of the search 'steps forward', moving through the formal plan of the database. Static scoping can be useful for finding the best possible match for a desired search, while Progressive scoping would generally by used to move through a large-scale musical form.

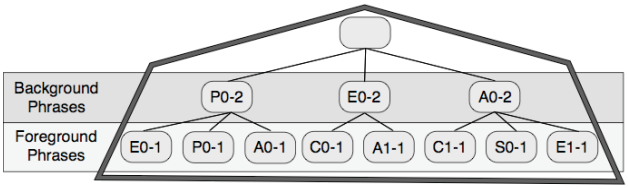

Figure 4. Static scoping of SPEAC hierarchy[2]

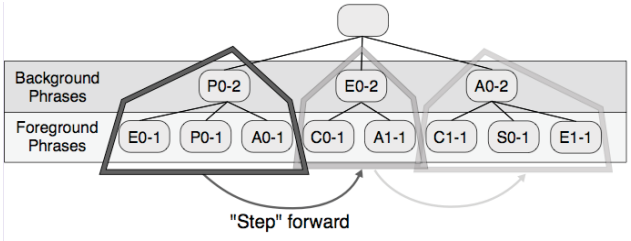

Figure 5. Progressive scoping of SPEAC hierarchy

Figure 6 demonstrates the way in which the MusicDB would handle searches with an input type of PitchList. It will be noticed that the actual data used for the return is always extracted in *relation* to the match found for the searched type. In the example, because PitchList is a field of the VoiceSegment object, the search for a best matching PitchList is carried out on VoiceSegments. From there, different return types (Chord, Contour, EDList) can be called, in relation to the VoiceSegment holding the match. In this case, the Contour and ED List are both stored by the VoiceSegment itself, whereas the Chord is stored by a Group, which is *referenced* by the VoiceSegment.

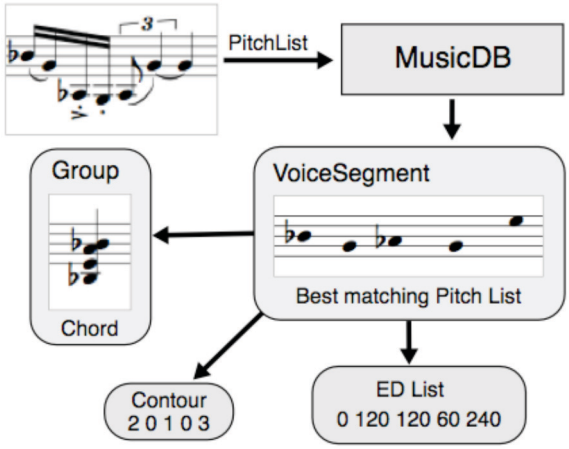

Figure 6. The system of references used to extract information from a given query

2.4. User Experience

Because input and output types are not required to match, the system offers great flexibility in the manner in which data is found and extracted. Take the following query as an example:

"PitchList 60 64 70 61 Current Accompaniment PitchContour Progressive"

This query would return all the Pitch Contours which accompanied the VoiceSegment that best matched the input Pitch List {60, 64, 70, 61}, given the current Scope, and would step 'forward' in the database once the search was returned. On the other hand, a query such as:

"PitchList 60 64 70 61 Following Element HarmonicMotive Static"

would search the *entire* database (due to the Static scoping) and return the harmonic sequence used by the Phrase *following* the VoiceSegment with the best matching Pitch List. Figure 7 shows a basic MaxMSP patch for querying the MusicDB.

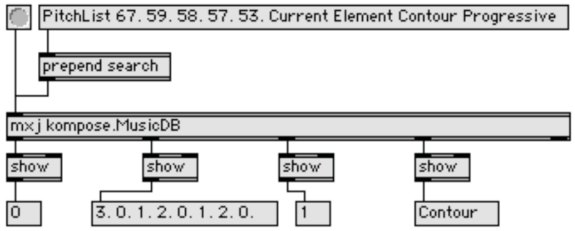

Figure 7. The MusicDB in MaxMSP. The second outlet reports the return data, the fourth return type, and outlets 1 and 3 provide Group count and voice number

[1] The branch chosen is dependent upon the last leaf to return a search. In the Figure 4 hierarchy, A0-1 would be followed by E0-2.

[2] This representation shows the tree-like pattern provided by SPEAC analysis. The label given to each node indicates the Phrase's formal function [9].

Composite searches are also possible, by entering multiple search queries *before* sending out the result. When queries are concatenated in this way, each query serves to narrow the scope of the search for the following query.

3. FUTURE DIRECTIONS

The MusicDB has been designed as a component for a larger system with the working title "ManuScore." This system will be a music notation-based application for interactive composition using principles of music recombinance. We will be working to extend the MusicDB itself by including the capacity for adding new material to the stored database dynamically, during the composition process. The other area we are working to develop concerns the discrimination of "quantized" versus "performed" rhythm. In particular, we are interested in refining the system in such a way as to make it more sensitive to the *differences* between quantized and performed rhythm, so that it may be given the capacity to apply a performed quality to otherwise quantized music, or to match the performance quality of fragments from two or more source works. One way we have thought to do this is by storing an event's ED as a *pair* of values, rather than a single value. The first value would indicate the quantized rhythmic position, and the second would indicate the offset of the performed rhythmic position from the quantized position. Analysis could be carried out using the quantized values, as in the current implementation, and the offsets could be used to reconstruct the rhythmic quality of the original performance. We are also considering the idea of integrating search categories more directly related to set theoretical analysis—perhaps a return type of "Set Similarity", for example—which could offer more flexibility in searching for materials based on specific pitch content.

Further, in order to introduce a more legitimate form of musical *inference* to the ManuScore system[1], we are also currently investigating the integration of Hierarchical Temporal Memory networks, modelled after those proposed by Jeff Hawkins, Dileep George, and Bobby Jaros at Numenta Inc. [9]. It is our feeling that the structural organization of the MusicDB could provide a useful pre-processing step in building the data representations used to train HTM networks, and could also provide category information for supervised training of the HTMs.

[1] Strictly speaking, the MusicDB is incapable of inference, being limited only to retrieving explicitly requested information from the data stored during analysis.

4. REFERENCES

[1] Assayag, G., Bloch, G., Chemellier, M., Cont, A., Dubnov, S. "OMax Brothers: a Dynamic Topology of Agents for Improvisation Learning", *Workshop on Audio and Music Computing for Multimedia, ACM Multimedia 2006*, Santa Barbara, 2006.

[2] Cadiz, R. "Fuzzy logic in the arts: applications in audiovisual composition and sound synthesis", *Fuzzy Information Processing Society*, Ann Arbor, Michigan, 2005.

[3] Cope, D. *New Directions in Music*, W. C. Brown, Dubuqe, Iowa, 1984.

[4] Cope, D. *Virtual Music*, MIT Press, Cambridge, MA, 2001.

[5] Cope, D. "Computer Analysis of Musical Allusions", *Computer Music Journal*, 27/1, 2003.

[6] Cope, D. *Computer Models of Musical Creativity*, MIT Press, Cambridge, MA, 2005.

[7] Dahlstedt, P., McBurney, P. "Musical Agents" *Leonardo,* 39 (5): 469-470, 2006.

[8] Eigenfeldt, A. "Drum Circle: Intelligent Agents in Max/MSP", Proceedings of the International Computer Music Conference, Copenhagen, Denmark, 2007.

[9] George, D. and Jaros, B. "The HTM Learning Algorithms", Numenta, Inc., Menlo Park, CA, 2007.

[10] Lerdahl, F. and Jackendoff, R. "On the Theory of Grouping and Meter", *The Musical Quarterly*, 67/4, 1981.

[11] Maxwell, J., Eigenfeldt, A. "The MusicDB: A Music Database Query System for Recombinance-based Composition in Max/MSP" Proceedings of the International Computer Music Conference, Belfast, Ireland, 2008.

[12] Morris, R. "New Directions in the Theory and Analysis of Musical Contour", *Music Theory Spectrum*, 15/2, 1993.

[13] Roederer, Juan G. *Introduction to the Physics and Psychophysics of Music*, Springer, New York, NY, 1973.