The MusicDB: A Music Database Query System for Recombinance-based Composition in Max/MSP

2 authors:

James Maxwell
Simon Fraser University
9 PUBLICATIONS **54** CITATIONS

SEE PROFILE

Arne Eigenfeldt
Simon Fraser University
74 PUBLICATIONS **647** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project   Generative Media View project

Project   Musebots View project

The MusicDB: A Music Database Query System for Recombinance-based Composition in Max/MSP

James B. Maxwell

jbmaxwel@sfu.ca

Arne Eigenfeldt

arne_e@sfu.ca

School for the Contemporary Arts
Simon Fraser University, Burnaby, B.C.
Canada

ABSTRACT

We propose a design and implementation for a music information database and query system, the MusicDB, which can be used for data-driven algorithmic composition. Inspired by David Cope's ideas surrounding composition by "music recombinance", 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

Techniques for algorithmic music composition can be roughly divided into the following prominent categories: rules-based methods [1], stochastic methods [10], data-driven methods [6], neural network models [13], fuzzy models [5], models inspired by biological and/or natural processes [2, 3, 4], and hybrid models[1]; the MusicDB is designed to function under the general category of data-driven methods. That is, data is extracted from the analysis of existing musical works, and can then be used as a knowledge base for the composition of new works. The MusicDB is not, itself, an algorithmic composition system, but is rather an analysis system, database, and search engine, designed to *facilitate* data-driven composition.

1.1. Music Recombinance

One of the most prominent figures in the field of data-driven music composition is David Cope [7, 8, 9], whose software "Experiments in Musical Intelligence" (EMI) became famous, even notorious, during the 1990s for composing vast numbers of highly convincing new works 'in the style of' Bach, Mozart, Beethoven, and Chopin, to name a few. The fundamental idea behind Cope's work with EMI was the notion of style emulation through *music recombinance*—the technique of creating new music through the atomisation and recombination of existing music. Cope suggests that style can be

analysed as a historical pattern of 'borrowings', embodied by the musical output of a composer, and carried out over the successive output of generations of composers, and that style emulation could thus be realised using a similar approach [9]. Cope developed a system of music analysis and data extraction[2], which we have used as a starting-point in the design of the MusicDB.

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 [14], kinesis [9], periodicity [11], 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 even be used as a parameter control module for an agent-based, or fuzzy logic-based, system—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 (Figure 1).

---

[1] Another prominent category is the rapidly expanding field of "agent-based" models and "Artificial-Life" (A-Life) models, which, for the purposes of this discussion, can be considered "hybrid", as they generally include techniques drawn from the above categories.

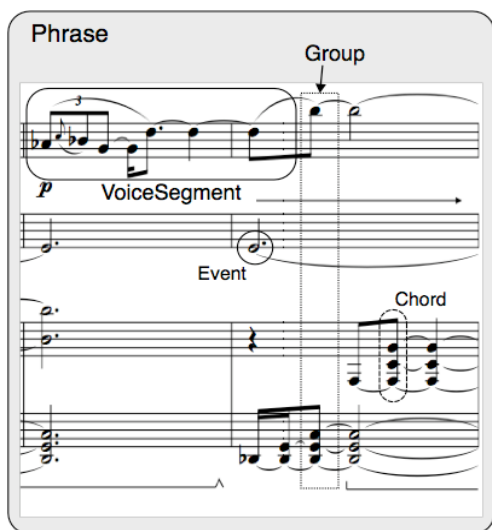[2] A full description of the Cope's "SPEAC Analysis" can be found in numerous publications [7, 8, 9].

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.

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 hold references to all VoiceSegments found to intersect their Group sequence.

The MusicDB uses Cope's technique of SPEAC analysis [9] to parse the horizontal aspects of a score into VoiceSegments and Phrases. The SPEAC system offers a simple, elegant, and surprisingly powerful method for extracting formal information from symbolic, or 'scored' music[4]. SPEAC can be used to generate a hierarchical analysis of musical form, making it easy to look at an analysed work from the level of individual Groups, Phrases, 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.

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. It is this interconnectivity by object referencing that is exploited by the MusicDB during the search process.

2.3. Query System

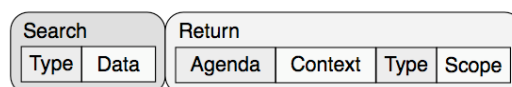Data is extracted from the MusicDB using a six-part query syntax (Figure 2).

Figure 2. Format of a search query.

Searches are carried out on VoiceSegment, Group, and Phrase objects, which hold the following fields[5]:

- **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

---

[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 adopt the frequently used 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, which will be shown later in the paper.
[4] Our implementation of the SPEAC system is well suited to the analysis of symbolic representations of music, not to audio recordings.

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

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 sorted set 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[1] (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.
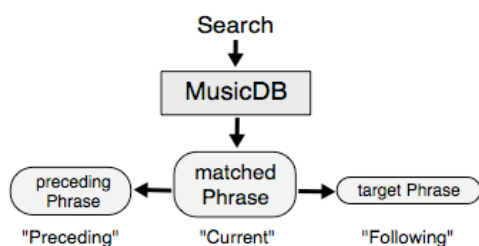

Figure 3. Search Agenda.

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[2]. If the requested field is held by a Group object, the *Group sequence* of that object's root Phrase will be returned[3]. 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[4]), 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 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[5], 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.
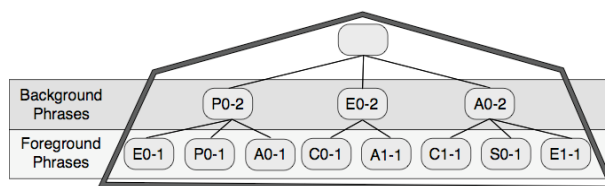

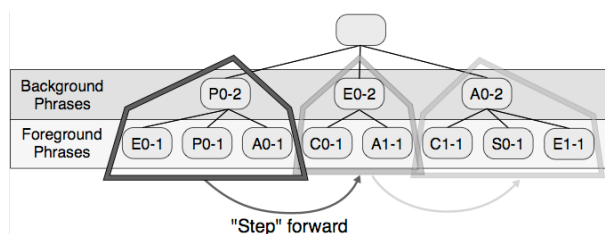**Figure 4.** Static scoping of SPEAC hierarchy[8].

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

---

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

[2] 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.

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

[4] 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..

[5] 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.

[8] This representation shows the tree-like structure provided by SPEAC analysis. The label on each node indicates the Phrase's formal function: **S**tatement, **P**reparation, **E**xtension, **A**ntecedent, **C**onsequent [9].

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.
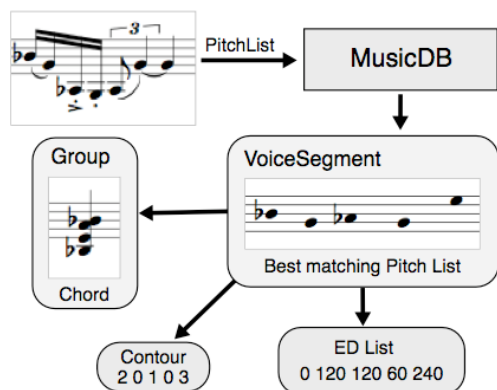

Figure 6. The system of references used to extract information from a given query.

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}, 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.
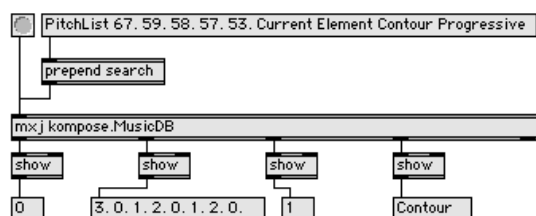

Figure 7. The MusicDB in MaxMSP. The second outlet reports the return data, the fourth return type, and outlets 1 and 3 provide Group count and voice number.

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 next 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 also be working to extend the MusicDB itself by including the capacity for adding new material to the stored database dynamically, during the composition process.

4. REFERENCES

[1] Ames, C. "The Markov Process as a Compositional Model: A Survey and Tutorial" *Leonardo*, 22/2, 1989.

[2] Beyls, P. "The Musical Universe of Cellular Automata", *Proceedings of the International Computer Music Conference*, Columbus, Ohio, 1989.

[3] Biles J. "GenJam: A Genetic Algorithm for Generating Jazz Solos", *Proceedings of the 1994 International Computer Music Conference*, Aarhus, Denmark 1994.

[4] Blackwell, T. M. and Bentley, P. J. "Improvised Music with Swarms", *Proceedings of the Congress On Evolutionary Computation*, Honolulu, USA, 2002.

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

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

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

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

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

[10] Jones, K. "Compositional Applications of Stochastic Processes", *Computer Music Journal*, 5/2, 1981.

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

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

[13] Mozer, M.C. "Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-scale Processing", *Connection Science*, 6/2-3, 1994.

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