Music Listening as Information Processing

Eliot Handelman[1] and Andie Sigler[1,2]

[1] Centre for Interdisciplinary Research in Music Media and Technology (CIRMMT)
[2] School of Computer Science, McGill University
andrea.sigler@mail.mcgill.ca

Abstract. Computational reasoning about musical structure (in particular, pattern, shape, and motion), with a perceptual and mathematical basis in simplicity.

Keywords: Musical structure, artificial intelligence, computer models, simplicity

1 An Information Processing Task

Paralleling a theory of vision (as stated e.g. by Marr [1]), it is possible to treat musical listening as an information processing task. In computational vision research, input is known as "I" for image, and the goal is to find functions that elucidate various properties of this image. The dual to this can be found in the (drawing-automaton) work of Harold Cohen [2], who asks "what is the minimum condition under which a set of marks functions as an image?" Both questions are also fundamental questions for music research, with the "images" in question being musical instead of visual.

Marr's three levels of description for an information processing task are the computational, the algorithmic, and the hardware. At the computational level, we can ask what is the *function* (or program) to be computed; at the algorithmic level we ask what are the *types* or the *language* in which such a function may be expressed; at the hardware level, we ask how functions in the language can be run on a physical computer (or brain).

This paper is focused at the algorithmic level, on the development of a few *types* for reasoning about musical structure.[3] The goal is to facilitate a higher level of structural description, so that further studies at the computational level (these may be statistical, musicological, generative, etc.) may be carried out on multi-dimensional complexes of pattern and shape, rather than on sequences of notes.

It is useful for a system of types to be *composable*. A composable system is one in which higher-level types can always be built from lower-level types

[3] These types have been implemented as part of a large-scale musical AI project, and can be explored through an interactive visualization system. Promising experiments have been made towards using types to direct "orchestrations" in order to sonify aspects of an analysis.

without having to define new types or composition functions at each new level. For example, given a type *list* and a function to put (any) things together into a list, we automatically get *lists of lists of lists of lists* and so on.

The types proposed for describing musical structure fall into the three categories of *pattern*, *shape*, and *motion*.[4] These types are composable, allowing us to speak of a pattern of patterns, a shaped and patterned motion, a pattern that has a shape, and so on.[5] With just a few definitions, therefore, it is possible to build from local structure up to the large scale.

In order for types to be composable, it is necessary that they be *rational* (i.e. logical or algebraic). Therefore, commonplace music-theoretic notions such as the labeling of harmonies, keys, or typical forms are avoided. Any system of labeling that is non-exhaustive is an *empirical* system, based on knowledge (and theorization) of existing music. The rational methodology we propose is exhaustive in the sense that we can't "look for" ABA patterns without also seeing AAA, ABB, ABC, $ABCD$, and so on. In the computational study of music, it makes sense for a rational level to come first. Later studies may impose or deduce an empirical level on top of the basic rational level, perhaps by constraining general types to fit a particular theory or answer a particular question, or by using statistical techniques on musical corpora.

Since questions of style and musical "language" are empirical, the utility of a system of rational types should not depend on the kind of music to which they're applied. At a rational level of structural analysis, we *do not need to know* the context or genre of the music in advance of the analysis.

What is required is a theorization of basic musical material and its possibilities. For example, we assume nothing of a melody but that it's a succession of pitches, each with a duration, and that pitch may be represented as a total ordering (i.e. for two pitches X and Y, there are exactly three options: X is higher than Y, Y is higher than X, or they are equal). Given an input "image" of this sort, what kinds of structures are inevitably found within?

1.1 Simplicity

On a strictly mathematical basis, there are many possible answers to the foregoing question. To narrow the field, we can ask "what kinds of structures might be interesting to explore?"

Since music is made by and for the human mind, an interesting analysis might recognize the kinds of structures that would be most *obvious* to a human: *simple* structures. Simplicity is interesting because in music, as elsewhere, the simplest things are the most salient (obvious), and certainly all music distinguishes within its discourses differing degrees of salience. Composing music implies managing saliences, since it is these that bubble to the surface of a piece of music, participating in its character. A structural musical analysis examining

[4] A fourth category, *texture*, has not yet been developed.
[5] As an example of a pattern with motion, consider $ABABBABBBABBBBABBBBB$, where the B segments display a (patterned) growth.

the management of saliences has a chance of revealing something about how music may be composed and how it may be heard.

Simplicity provides a perceptual basis for music analysis; it also serves as an effective base case for reasoning, allowing a systematic analysis of basic musical material. While we cannot speak of maximally *complex* music, the *simplest* ways to make a piece of music can be enumerated.

Likewise, the simplest ways to make a pattern, a shape, or a motion can be enumerated. The simplest pattern is just repetition of a single term: $AAAAAA$.... The simplest shapes (in a total ordering such as pitch, or loudness, or set cardinality) are those that are described by just one orientation: *UP* or *DOWN* or *SAME*. Motion, as well, is simplest when it proceeds in just one direction.

1.2 Avoiding the Encoding Problem

The questions that follow are somewhat trickier: what is the *next simplest* pattern? what are the *next simplest* shapes? And what are the next simplest patterns and shapes after those?

It is tempting to devise a system to compose a few simple types such that (for example) *any* pattern can be encoded under the system.[6] After some initial exploration, we decided to avoid the "encoding problem," for the reasons that follow.

Under any coding system, there may be several ways of encoding a given pattern. We then must ask which encoding to prefer. A common solution is to prefer the shortest encoding, since it compresses the pattern as much as possible, thus making use of as much of the pattern's inherent structure as possible. However, the search for a shortest encoding is computationally hard (i.e. no efficient, deterministic algorithm is known).[7]

Even if we do manage to find (or approximate) a shortest encoding, we are left with a *measure* of pattern complexity (or entropy), and a single way of maximally compressing the pattern. Our problem, however, was not to *compress* the pattern, but to *describe* it in such a way that simplicities are indicated. This may turn out to take up *more* bits than the original pattern, since there may be many interesting things to point out which cannot all be summarized in a single encoding.

We may instead ask for a few different encodings, or all possible encodings. But this is still computationally hard, and still of limited utility for our problem. Imagine we have a pattern that looks like alphabet soup, but every time the term X appears, it is in a group of Xs such that each group is one term shorter than the last. Depending on our coding system and the algorithm we use for finding encodings, this regularity may or may not be expressed somewhere in our results. But even if it is, it will be located in some encoding of the entire pattern, and

[6] Something like this was attempted by [3], and much work was done along these lines by structural information theorists [4, 5].

[7] A non-deterministic, genetic approach to this problem is explored by [6].

we will still have the task ahead of us of searching through our encodings for simple patterns.

An alternative approach is to avoid the encoding of non-simple patterns, and instead to focus on finding simple structures within them. In the alphabet-soup example above, the system might not be able to provide a concise description of the entire pattern, but it should be able to point out that in the X dimension, at least, something simple is happening.

The total "dimensionality" of a piece of music is very large, and may be impossible to enumerate. A structural description therefore can't claim to be exhaustive. Instead, general methods are developed to unyoke structures into separate dimensions and to synchronize dimensions into higher-order dimensions. The separation and resynchronization may be directed by a higher-level program, randomized, guided by the simplicities discovered, or dictated by a combination of these methods. The composable type system allows any number of dimensions of pattern, shape, and motion to be explored without requiring the definition of new types to represent them, or of new functions for discovering and analyzing them.

In the remainder of this paper, we discuss a few primitive types and means of combination for musical shape, pattern, and motion. Because space is limited and work is ongoing, the description provided here is not of a complete system.

2 Shape

Shape can be built on the basis of *orientation*. Oriented shapes occur everywhere in every oriented (or quantifiable) aspect of music, including pitch, loudness, speed, density, and so on.

A simple shape called a *chain* is described by just one orientation, *UP*, *DOWN*, or *SAME*. Given a melody, it is easy to decompose its pitch content into a sequence of chains, with each chain overlapping the next by one pitch.

The *chain* is a simple type describing a local structure. It is recursively composable to describe large-scale structure. Recursive chains are called *Z-chains*.[8]

2.1 Recursive Orientation: Z-chains

First-order Z-chains (or Z-chains described by one orientation), are just chains. The first step in the composition of chains into second-order Z-chains is the unyoking of the three orientations into three dimensions. Thus, when chains are "chained together" to form higher-order chains, the principle of *chaining like to like* is observed.

Chains (and Z-chains) can be compared with respect to several different oriented features, including top pitch (more generally, top value), bottom pitch, chain length, and interval span. Z-chains are found in one feature at a time.

Having specified a feature and a sequence of nth order Z-chains in one (recursive) orientation, the same simple one-pass algorithm that finds chains is used

[8] The Z stands for the zig-zagging shapes that result.

to find Z-chains of the $(n+1)$th order. For example, given a sequence of chains going *UP* and comparing their top pitches, the chaining algorithm decomposes the sequence into Z-chains (in feature top pitch) with orientations *UP-UP*, *UP-DOWN*, and *UP-SAME*. The recursive Z-chain algorithm unyokes these new dimensions, and runs the chaining algorithm again on each.

Each pass of the chaining algorithm is shorter than the last, since at every subsequent level there fewer chains. The algorithm terminates when it finds only one chain of a given recursive orientation, since there remains nothing to chain it to. The algorithm for finding all Z-chains in a sequence (for a given feature) is efficient, taking time $O(n^2)$ where n is the number of items in the sequence.

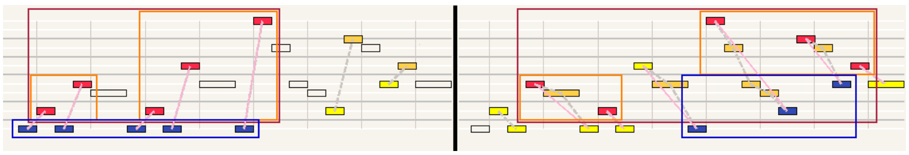

Fig. 1. Z-chains in "Happy Birthday." On the left, top pitches UP-UP-UP and bottom pitches UP-SAME; on the right, tops DOWN-DOWN-UP and bottoms DOWN-UP.

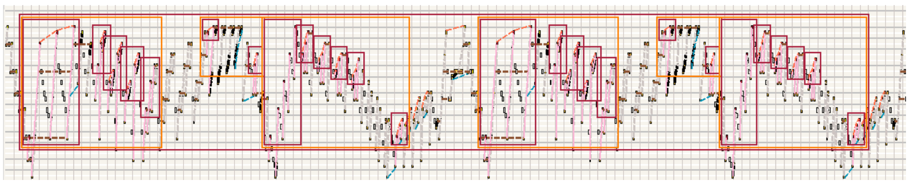

Fig. 2. A large scale fourth-order Z-chain in the first part of *Presto* from J. S. Bach, solo violin Sonata I. The outer box encapsulates the repetition.

Figures 1 and 2 illustrate the Z-chain concept over small and large scales, respectively.

Higher-order Z-chains may skip items in the sequence – in particular they skip Z-chains in orientations other than their own. This property allows us to infer that the orientation in question *does not exist* in a skipped stretch of music. For a long piece of music, very high-order Z-chains may provide an overview of some aspects of form, but they may also be fragmentary. Z-chains of second and third orders, on the other hand, tend to provide compact synopses of short stretches of music.

Z-chains are based on the concept of chaining *like to like*. A common motivation in computational music analysis is to search for *repetition*. This is often done by comparing over note n-grams, possibly with a tolerance for error to admit variation. The Z-chain scheme of chaining like to like, on the other hand, is an efficient search for *parallelism*, allowing similar structures (for a strictly-defined definition of "similar", *not* an error tolerance) to be linked. The link is itself *oriented*, so that instead of saying of a structure "here it is, and here it is again" (as in the search for repetition), a higher-level structure is constructed in which the component lower-level structures stand in a specified oriented relation to one another.

2.2 Synchronizing Z-chains

A synchronization function is a means of combining structures in different (orthogonal) dimensions. *Mini-schemas* are made from inclusions (overlaps) of Z-chains in different orientable features (i.e. top pitch, bottom pitch, chain length, etc.). Mini-schemas are specified by Z-chains in at least two different features, and instantiated (as Z-chains are) as lists of chains.

Mini-schemas are constrained to consist of an *entire* Z-chain of some order in (at least) one feature, and to consist of *consecutive* first-order chains. Given a feature-set minimally including "top pitch" and "bottom pitch," the union of all mini-schemas always covers the entire sequence, since every two consecutive pitch chains have oriented top pitches and bottom pitches.

A mini-schema specification may be instantiated more than once in a given piece. The multiple instances of a schema need not be note-for-note identical, they are only "the same" with respect to their multi-feature Z description (and their compactness with regards to chains). They need not be the same in all features, but only in the subset of features which is used in their description. They need not contain the same number of first-order chains.

If there is more than one instance of a given mini-schema, it is possible to compare instances to discover how the schema is deployed throughout the course of the piece.

Mini-schemas are illustrated in Figure 3.

3 Pattern

A fact of music is that difference is valued: composers are obliged to produce music that does not repeat *any* known music. It is also true that within a single piece, differentiation is important, since it is difference that articulates form (at all scales) and through which discourses occur.

In contrast with oriented shape, in which any two terms are comparable within a total ordering, a comparison of *pattern* terms tells us only whether they are equal or unequal. In a pattern like ABA, nothing is known about the relationship between A and B, except that they are different, while A and A are the same.

An obvious mode of pattern analysis is the detection of recurrent subsequences, for which well-known methods apply. A second mode involves the recognition of "concentric" patterns, in which the ABA pattern is generalized by considering B as an *island* in a sea of A. Similarly, the following pattern has an island of C: *(ABBA(CC)ABABAA)*.

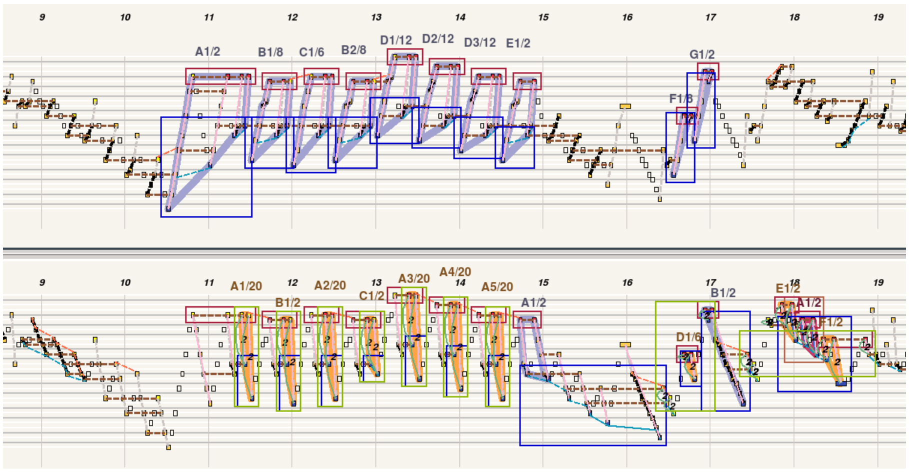

Fig. 3. Schemas in J. S. Bach: *Gigue*, Solo violin Partita II. Top half is view of chains UP, bottom half is chains DOWN. The schemas outlined in lilac in the top half are defined by constant pitch top, rising pitch bottom. The orange schemas in the lower view are "hotter", comprised of the additional feature of constant chain length, shown with green arcs.

The concentric-pattern algorithm has, as a natural consequence, the effect of segmenting stand-alone sections such as *(ABCABCABCABC)(DEFDEF)*, in which no terms in any top level group occur in any other top level group.

It is possible to generate patterns from schemas by comparing instances of a schema specification under an equality predicate (e.g. exact equality, equality under transposition, etc.). Among these are sure to be some easily identified as "canonical," fully interpretable as structures of repetition within concentricities. The segmentation of a Bach work composed of two repeating sections proves to be a trivial consequence of this analysis.

4 Motion

Motion naturally leads us to inquire into rhythm, but here we restrict ourselves to the motion of schemas. Schemas allow for the construction of "motion shapes" in the following way. Schemas are partial descriptions: nothing precludes a description involving three features partially covering a description with just two features of the three. We may presume that the schema with more synchronized features is more regular than one with fewer: in McLuhan's (jazz-influenced) terminology, the more regular schema is the "hotter" pattern, where a "hot pattern" is said to "drive" perception and a "cool" pattern is said to invite perceptual completion. The partial covering of a "cool" pattern – involving few features – by a "hotter" pattern – a superset of the cool pattern – can be thought of as a heat transition. Figure 4 shows an example.

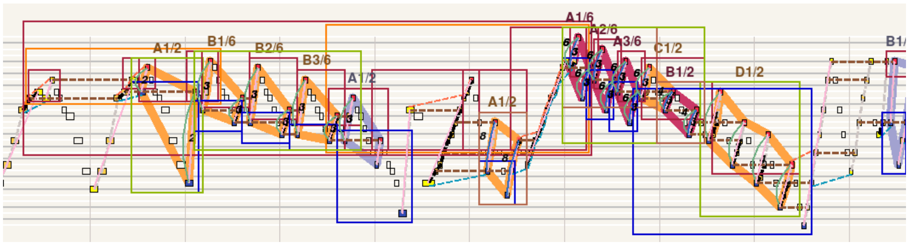

Fig. 4. *Gigue*, near the opening. Differential heat in the same schema, comprising *heat transitions*. Red is hottest, i.e most regular, followed by orange and lilac. The boxes show the Z-chains synchronized by the schemas.

5 Reasoning About Music

One benefit of the foregoing system lies in its ability to bring out formal structure in arbitrarily simple or complex music, constructing divisions based on pattern-recurrence of schemas. A further goal is a much more extensive elaboration showing the interrelation of the parts of music.

The aim, in the material presented, was the generation of a "primary" (and deterministic) level of musical objects. More objects are constructed by discovering further relations, generating more shapes and patterns, at which point the system is fully re-entrant. For example, each instance of a schema yields a pattern of *length* in its base chains: the pattern can be treated as a Z-chain, whose synchronicity with any other features can be schematized. From this point on, analysis may proceed *non-deterministically*, since object generation is potentially unlimited, and the possibilities of synchronization are vast.

References

1. Marr, D.: Vision: A computational investigation into the human representation and processing of visual information. W. H. Freeman, San Francisco (1982)
2. Cohen, H.: The further exploits of Aaron, painter. Stanford Humanities Review, 4.2 (1995)
3. Simon, H.A., Sumner, R.K.: Patterns in music. In: B. Kleinmuntz (Ed.) Formal representation of human judgement. Wiley, New York (1968)
4. Leeuwenberg, E.L.J.: A perceptual coding language for visual and auditory patterns. American Journal of Psychology, 84, 307-349 (1971)
5. van der Helm, P.A.: Cognitive architecture of perceptual organization: From neurons to gnosons. Cognitive Processing, 13, 13-40 (2012)
6. Dastani, M., Marchiori, E., Voorn, R.: Finding Perceived Pattern Structures using Genetic Programming. In: Genetic and Evolutionary Computation Conference (2001)