Pattern Play

Declan Murphy*

Music Informatics Group
Computer Science Dept
Copenhagen University
declan@diku.dk

Abstract. This paper presents a progress report on a system to facilitate the manipulation of structure in music. Its purpose is to serve as an infra-structure for exploring music in terms of its inherent pattern structure, providing a platform for further research into the perceptibility of musical pattern. It is also intended as a framework for composition and, ultimately, as a performable instrument.
Structure is represented in terms of patterns extracted from features, which may then be rearranged via a graphical and gestural user interface. Aspects from a given passage of music can be imposed upon, and mixed with, other sections of music. There is also the facility to capture expressive gestural user input: both for immediate generation of phrases and for imposing phrasing interpretation onto passages.

1 Introduction

A key result of psychoacoustics is that the *meaning* in a piece of music arises – to a very large extent – out of our perception of patterns in the piece (see, for example, [1, ch. 5], [2] or [3]). Certainly there are various other factors (such as timbre and social context) contributing to the meaning of a piece of music, but perception of pattern is perhaps the most abstract and pervasive of all.

The more modern techniques of analysis in musicology aim at an objective analysis of a piece in terms of its structure, using techniques which are well suited to computerisation. This system, *Pattern Play*, implements some of these techniques to represent music in terms of its inherent pattern structure.

Eugene Narmour's Implication-Realisation model of melodic structure, [4] [5], shows how melodies can be parsed into five atomic 'primary archetypes' (such as AA, AB, ABA′) and associates these with Gestalt psychology grouping principles. Lerdahl and Jackendoff have another approach, recursively partitioning a piece according to four kinds of structural analyses, based on the listener's expectations ([6] [7] [8]). Pierre-Yves Rolland introduces the *FlExPat* system, [9], to allow user control over the kinds of patterns extracted. This system proposes to provide a platform for further investigation of these issues.

* Components of this work were carried out while the author was a visiting researcher at Århus University, at the CART lab of CNR Pisa, and at the InfoMus lab at the University of Genoa.

This internal pattern representation may be input to the system either by an analysis of a piece or passage supplied by the user, and/or by use of the Graphical and Gestural User Interface (GGUI).

Once a pattern representation exists, it may then be manipulated via the GGUI. Such manipulations may include:

- high-level re-arranging of the piece's structure,
- imposing abstract structural patterns or local phrasings from one piece onto a new piece, or
- beating out rhythms, describing melodic contours or phrasing interpretation by expressive hand gesture.

The gesture interface is used both for control of the pattern manipulation and for expressive gesture capture.

The system includes a feature extractor and analysis engine, native score and feature pattern formats, an infrastructure for composing and rearranging music based on its pattern representation, a graphical user interface for display of pattern representations, and a gestural user interface for manipulation of these patterns and for capture of expressive gesture. Conceptually, the system may be represented as in figure 1.

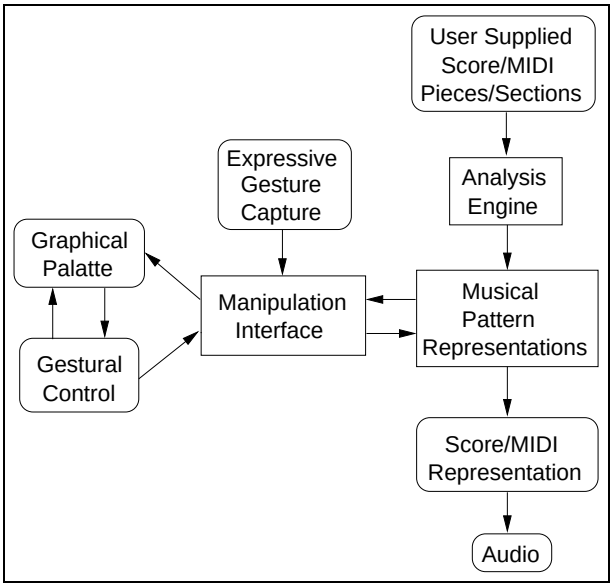


Fig. 1. System Conceptual Schematic

Section 2 describes the internal score representation, and the native structural representation. §3 outlines the analysis algorithm. §4 describes how the system is implemented, §5 describes the user interface, and §6 concludes with the state of development.

2 Representation

2.1 Background

By and large, music from any part of the world, from classical to popular, can be neatly analysed in terms of rhythm and of harmony, with melody having aspects of both. A good survey of music representation issues can be found in [10].

Within musicology, *analysis* is quite a formal and well established subject (see [11] for a good introduction and authoritative overview of the subject). While many of the techniques of musical analysis begin with subjective considerations, some of the more recent ones are more objective and are particularly suited to computational methods. These latter include statistics of musical features, information theory, semiotics, linguistics, and equivalence classes of chords. For this paper, we consider pattern recognition (in the generic sense) as applied to music.

However, most of the work in the field of pattern recognition is of an open-ended stochastic nature, whereas in music there is generally some kind of resolution involved. A coherent piece of music always has aspects of confirming the listener's expectations, along with aspects of deviating from them. (This, indeed, is the motivation underlying the above-mentioned information theory school of analysis.) These deviations, however, almost invariably return to a sense of 'home' or origin, having played out some kind of tension in the journey. In this light, a more closed, discrete, approach to pattern recognition is considered more appropriate for musical patterns.

Moreover, most of the existing applications of the field of pattern recognition are aimed at the problem of trying to recognise the existence, location and orientation of objects in real-world data (typically computer vision) whose attributes are known beforehand. There is normally a degree of uncertainty involved. The input to this system, however, is precise data (note pitches, timings, dynamics, etc.): the problem is to represent it in terms of its implicit higher level pattern structure.

Simon and Sumner, [12], pioneered a technique to formally represent patterns in music and outlined how this could be automated. The analysis pursued here is at a slightly lower level, more focused on the data *per se* than with higher level musical cues.

2.2 Input Format

Rationale. In musicological analysis, the score is taken as the starting point and standard reference frame, and for good reason. While audio would be more complete, a trained musician has no difficulty in making the transition from score to realisation (with appropriate phrasing, etc.) in their imagination. Audio representations (digital recordings, spectrograms) have too much information and are relatively cumbersome to navigate; the score has evolved as a concise random access representation of the higher level musical structure from which the music may be reproduced.

Also, from a practical point of view, a score level representation is simple and accurate to deal with computationally. (Accurate segmentation of even monophonic melodies from audio files is far from straightforward (see, for example, [13]).) Nonetheless, micro-level pitch and timing information is indeed included to facilitate expressive phrasing and audio output.

It could be argued that contemporary music has a greater focus on timbre, while adopting an iconoclastic attitude towards harmony and form as compared with classical music theory, and that the approach taken here is therefore of diminishing relevance to today's composers. However, a *pattern* is a highly abstract notion, applicable far beyond the realm of melodies[1]. Indeed in contemporary music, the focus on musical structure is as strong as ever, often perhaps more so, so that an abstract platform for structural representation and manipulation seems timely.

Having said that, however, we start out by considering a monophonic melody under the classical western theory of harmony. Consideration of more modern, looser musical structure will follow.

Score Representation. The system has its own internal representation (i.e. a file format) for musical input, manipulation and output. Routines are provided for direct conversion of MIDI files (amongst other Music V compatible formats) into this format, and for conversion from this native score to musician-readable score, MIDI and audio for system output.

The native score file format has all the advantages of MIDI files (in that they carry a minimum of information which can be directly played, yet enough low level pitch and timing information to carry expressive phrasing), but retain all the features of the usual score notation (phrasing marks, key and time signatures, bar lines, explicit dynamic directives, etc.). (In fact, only redundant score attributes, purely for the benefit of the human reader and wholly unnecessary for a computer are formally absent. These include such attributes as key/clef change prompts at the end of lines, octaviation notation, ledger lines, etc. These markings can however be added afterwards if a readable score output is desired.)

2.3 Pattern Representation

Features. The following musical features are first extracted from the score representation:

1. note rhythm intervals (considered as a string of independent objects)
2. note pitch levels (considered as above)
3. rhythm patterns (considered across the entire section, at all intervals)
4. pitch contour (considering relative up/down progression, no. of scale steps)
5. chord progressions

[1] Consider that numbers, ubiquitous as they are, are just one type of pattern. Gestalt psychology (e.g. [14], [15]) has much to say on how our experience of the world is a result of our perception of patterns.

6. note degree in chord (tonic, supertonic, etc. of underlying chord)

Thus, features 1,3 are of rhythm, 2,4 are of pitch, and 5,6 are of harmony.

Features 1,2 examine the data purely as a string of information, without heed to any of the relationships or degrees of similarity between them (i.e. equality is the only criterion). Rhythm data has the strict[2] ordering of a time sequence and interval values which are nominally simple whole number multiples of each other, and so we subsequently consider the resulting higher level inter-relationships with feature 3. Pitch data has the strict[2] ordering of higher/lower frequency with perceivable steps, and so these melodic contour patterns are extracted from feature 4.

Even without explicit accompaniment, monophonic melodies clearly[3] describe chord progressions – hence feature 5 – with the melody tracing its way through them: how exactly it does this gives rise to feature 6.

Analysis. First, each feature is extracted and subjected to its own pattern analysis.

Much of the magic of music, however, arises out of the way in which multiple levels of patterns occur simultaneously. For example, at a basic level, every note has both duration and pitch, confirming and/or varying the rhythm and harmonic expectations every time. There are also higher level (melodic structure) and lower level (phrasing and timbre) multiplicities of greater subtlety and complexity. The approach of this system considers this parallel multiplicity essential to musical patterns.

It is in this multiplicity that the analysis algorithms used differ from compression algorithms: both seek to find neater ways in which to represent the data, but compression is only interested in a single optimum strategy whereas here all compressions above a certain entropy threshold are considered interesting (perceivable) and are recorded.

On top of this, the final stage of analysis is a cross-feature pattern extraction.

The analysis output has its own notation, briefly summarised below. It is hoped that the following simple example will serve to illustrate the main ideas.

Notation. All data are of the form of a *string* (finite sequence) of characters from an *alphabet* (finite set), Σ. Let Σ^* represent the set of all possible strings, so that we may write $\sigma \in \Sigma^*$ for a string σ. Let $\sigma = "\sigma_1\sigma_2\cdots\sigma_n"$, $\sigma_i \in \Sigma$. Concatenation is denoted by juxtaposition, so that $\sigma\sigma = "\sigma_1\sigma_2\cdots\sigma_n\sigma_1\sigma_2\cdots\sigma_n"$. We write 2σ to mean $\sigma\sigma$, and similarly for $i\sigma$, $i \in \mathbb{N}$. We define a *string cycle* by

$$\rho : \mathbb{N} \to \Sigma^* \text{ given } \rho(i) = \rho(i \bmod n) \text{ for all } i \in \mathbb{N} \text{ and some } n \in \mathbb{N}.$$

[2] Remember that we are considering only monophonic melodies here.

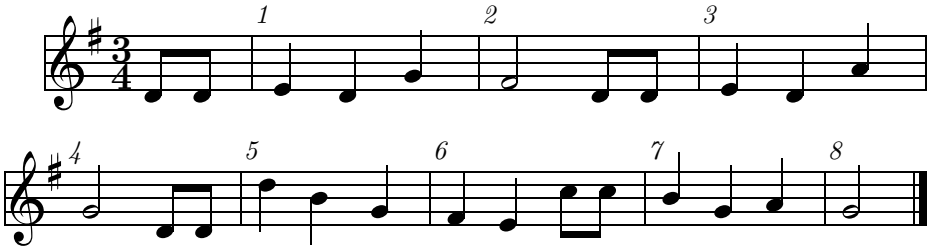
[3] Which exact chord a single note belongs to may, however, be ambiguous, and this is treated as the note belonging to each chord.

On the stave. In order to show string cycles on the stave, it is convenient to introduce a comma notation. Let $\sigma^1, \sigma^2, \sigma^3 \in \Sigma^*$. We may write

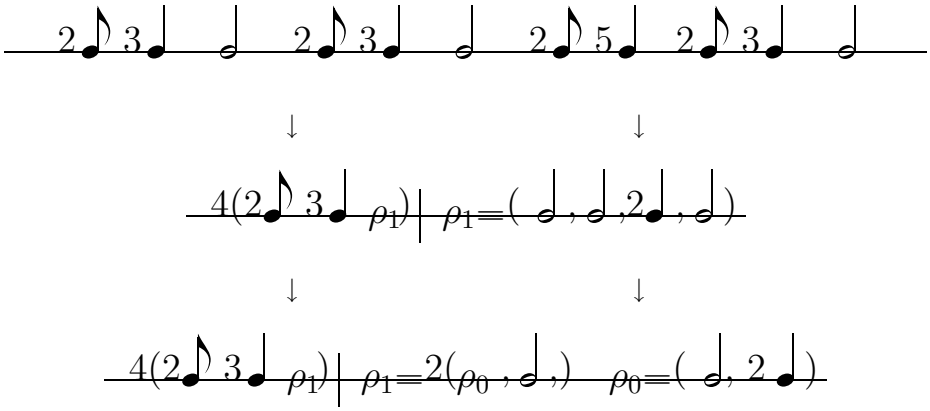
$$\rho = (\sigma^1\sigma^2, \sigma^3) \text{ meaning that } \rho(1) = \sigma^1\sigma^2,\ \rho(2) = \sigma^3,\ \rho(3) = \sigma^1\sigma^2,\ \text{etc.}$$

The section is reduced to the expression preceding the vertical (bar) line. Subsequent expressions expand this first compressed version. A short vertical bar is used to denote the place where a real bar line would materialise.

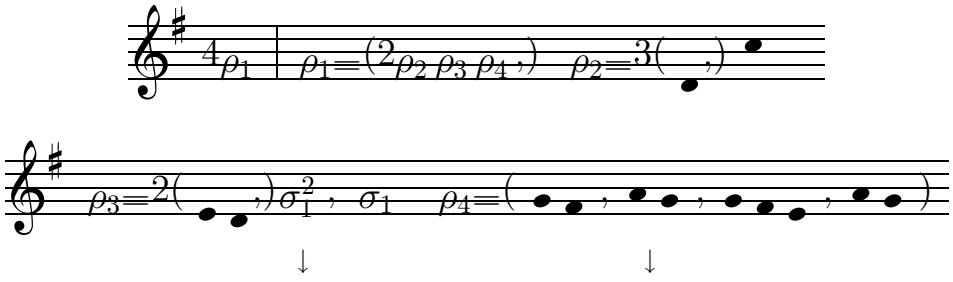
Example. The sample for analysis is the universal *Happy Birthday* song:

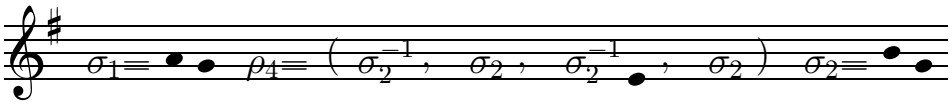


Feature 1 pattern analysis goes as follows:

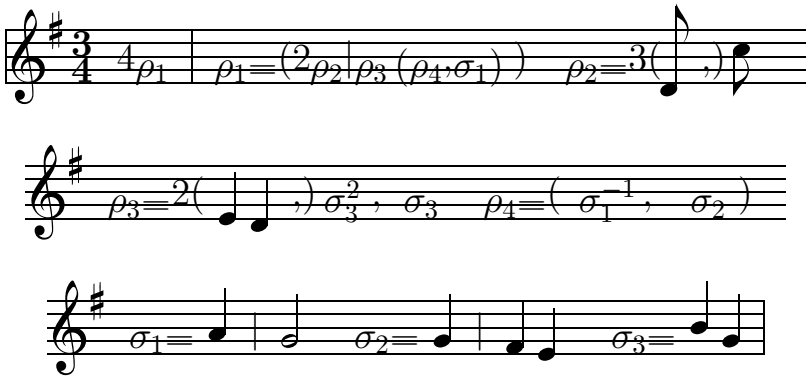


An instance of feature 2 pattern analysis produces the following (there are other possibilities):





Combining these features 1,2 gives:



3 Pattern Extraction Algorithm

3.1 Defining the problem

The entire analysis is being formalised mathematically, algorithmically and in hard computer code. An outline of the algorithm follows. The task may be stated as: *given a string and an associated entropy limiting criterion as input, we require all reduced representations of the input string up to the given limit as output.*

The entropy limiting criterion could be of the form "the representation with the lowest entropy (i.e. the most compressed)" or "all representations below a certain length" or "the n most compressed representations". The *entropy* of a string may be defined to be the length of the most compressed representation, according to Chaitin's algorithmic information theory [16].

Properly speaking, it is the theoretical minimum length of a program or input stream to a universal Turing machine which produces the string in question as output, which determines its entropy. A reasonable analogy (for present purposes) may be drawn between taking the minimum input to a universal Turing machine and taking the minimum output from the analysis routine presented here. It should be noted that the entropy of the input string to this analysis algorithm is absolutely fixed and should not be confused with the length of differing output representations for the string. We can, however, consider compressed representations of the same string of greater length to have greater redundancy and therefore, by reverse analogy, more "entropy". Intuitively, strings which can be highly compressed are "highly patterned", and vice-versa.

3.2 Outline Approach

Essentially the algorithm refines the input string $\sigma \in \Sigma^*$ in two stages: extracting the elemental substrings $E(\sigma)$ of σ, and then parsing σ in terms of $E(\sigma)$. This process is then repeated recursively.

The first routine is to extract the elemental substrings $E(\sigma)$. The wildcard '$\cdot$' is introduced which matches any character of Σ or the null character λ, and the wildnumber '$\dot{n}$' which matches any number $n = 0, 1, 2, \ldots$

1. Remove all duplicate members from a copy of σ, making it a list of singleton substrings. Call this list L_1.
2. Add all pairwise concatenations of members of L_1 to a copy of L_1. Call it L_2.
3. If this is the first call to E i.e., the zeroth level of recursion, then return L_2 and stop.
4. For all pairs from L_1 sharing a common member with another pair or with a singleton from L_1, add the concatenation of the common member with $\cdot$ as a member to L_2 preserving the order of the common member. E.g. if L_1 contains both "c_1c_2" and either "c_1c_3" or "c_1", then "$c_1\cdot$" should be added. Similarly, if L_1 contains 3"c_1" and 3"c_2" then 3"$\cdot$" should be added, and if L_1 contains 3"c_1" and 5"c_1" then $\dot{n}$"c_1" is added. From the second level of recursion onwards, the terms 'pair' and 'singleton' generalise to 'member' but matching is only necessary on the outermost characters.
5. Add "$\cdot$" to L_2 and return it.

The next routine is to parse σ in terms of $E(\sigma)$. By contrast with the previous routine, this one returns multiple 'parsings'. Strictly speaking it does not really parse in the usual sense since there is no single fixed grammar.

1. For all $s \in E(\sigma)$, locate all matches of s in σ. Where matches overlap, spawn multiple parsings and continue through the following steps for each one.
2. Form a parsing by concatenating members of σ by replacing them with members of $E(\sigma)$.
3. Perform a run length encoding of the parsing, i.e. replacing n consecutive occurrences of c by n"c" for $n \in \mathbb{N}, c \in \Sigma$.
4. Extract string cycles from instances of the wildcard '$\cdot$', and attach them to the parsing.

Finally the two routines operate in tandem recursively taking the parsings from the second routine as input strings to the first, terminating when the entire parsing/representation is in terms of just one element or when the entropy criterion is exceeded.

Various optimizations are still under investigation, such as the best way to calculate the length of the compressed representation for the entropy criterion; various early bail-out heuristics for skipping families of patterns which are going to be longer than others in the parsing routine; various heuristics for obtaining the neatest representations first, such as considering the longer strings in $E(\sigma)$ first; the degree of implementation of regular expressions, a reverse operator, a

permutation operator and a subsequence extractor (balancing the cost of finding them versus their perceptibility); flagging pairs for reverse so as to process only combinations instead of all permutations, etc. Although the efficiency of this pattern extraction algorithm is an issue, it is not necessary that it runs in real-time. The alphabet size for musical data is very small in practice, so that the only serious load comes from long passages. The latest version can be downloaded from [17].

4 Implementation

The prototype is being developed on a laptop computer running Linux. Two USB WebCams are used for gestural input, and a MIDI keyboard is attached for ready input of musical ideas.

All the high level programming (concerning music input, analysis, manipulation and output) is done in Common Lisp. The main advantages of Lisp are its particular suitability for dealing with such abstract concepts, and the ability to incorporate other available Lisp software as desirable components into the system.

The underlying protocol for user input and output is Common Music (CM) (available from [18]) into which the native score format (§2.2) is interfaced using the Common Lisp Object System (CLOS, the object oriented system of Common Lisp). Thus all the features of the Stanford (CCRMA) suite of (Lisp based) software for algorithmic composition, score notation and software synthesis are immediately and seamlessly available. Included in the interface are routines for converting to and from readable score notation (CMN) and MIDI files.

As well as being a useful common framework for processing and converting between various computer music formats, CM is primarily a tool for algorithmic composition. Thus, any material generated in CM is already in the system.

In order to intelligently extract rhythm values from input MIDI files and gestural rhythms, the Lisp routines[19] from the *Music, Mind, Machine* group at Nijmegen are also seamlessly incorporated. These are based on their work, [20], on modelling human perception of rhythm (which is often surprising!).

Although the facilities for data processing in Common Lisp are highly developed, the system also makes occasional use of the AWK language from Unix (viz. GAWK from the GNU/Linux operating system) which is specialised for extracting string patterns with regular expressions and for common processing operations on them. It is not clear at present if this will remain as a feature of the final system.

5 User Interface

The user interface is used for display of analysis results, structural rearrangement, and gestural input.

5.1 Graphical User Interface

Once some section(s) of music has been input, it is analysed as per §2.3 and these results are displayed. There are a number of different visual representations of the pattern structure under development. One is the readable analysis score as in §2.3. Another approach is a three dimensional view of geometric shapes arranged in an analogous fashion to the pattern structure. (There are many ways to do this, and some degree of choice is presented to the user.)

Controls common to all the graphical representations include selecting:

- the features of interest
- the desired level of analysis
- which among several valid representations is of interest
- which features and levels are taken for cross analysis
- the entropy threshold

Once the user gains a feel for the structure present, the next stage is to be able to play with it.

5.2 Gestural User Interface

Devices used for gestural input are:

- Mouse (for prototyping, may not feature in the final system),
- Dual WebCams, functioning as:
  - Baton Tracker,
  - Hand Posture Tracker,
- MIDI keyboard.

The hand posture tracker is based on a geometrical technique for reconciling multiple camera images (after appropriate pre-processing) to a physical model of the hand. It is a further development of the work of [21], written in C++.

The use of the gesture user interface is two-fold:

- as a means to manipulate the structure as displayed, and
- as raw input of rhythm, pitch and expression criteria.

In particular, the work of the group at KTH, Stockholm, [22], in mapping motion to musical parameters, is being incorporated via their expression rules. Also, some results from the EyesWeb[23] project are likely to be included.

5.3 Manipulation, Inference Rules, Output

The interface from expressive gesture to structural representation is in the form of an inference rule system. The *expression* (e.g. accentuation or timing), is coded in terms of a set of minimal requirements (deviations from the ambient dynamic level, micro level time information) on the native score format. Similarly, it is the representation of structure in terms of the minimum requirement rule system,

and the ability to infer from logical combinations of these rules, that facilitates the restructuring of a given section. Changing any parameter at any level of a pattern representation is propagated throughout the structure (the reverse of the analysis process). Taking structural requirements from one section and imposing them onto another creates something entirely new, but with properties of each.

As the internal score representation is in the form of CM objects, it is straight forward to get output in the form of MIDI, a musician readable score (via CMN converting from the native score format), or audio (via CLM or Csound if the user specifies such instruments).

6 Conclusion

6.1 Further Developments

The system will be used to investigate the relative perceptibility of various patterns, and this information will be fed back into the system as a criterion in the analysis affecting the thresholding as well as the entropy. The system is also being monitored in connection with research into requirements for composition tools.

The pattern extraction analysis will be refined to cope with large amounts of data in a reasonable amount of time. Other features for analysis, may be added (or those existing may be modified) as seems most appropriate with experience. Other suitable structural analysis techniques, such as parsing a piece in terms of its implicit grammar, may be added.

The system should be generalised from monophonic melody to full polyphony, spectral envelopes, and arbitrary abstract structure. These latter, however, are likely to require more technical input from the user.

6.2 Summary

This paper is a progress report on the *Pattern Play* system to explore the presented analysis technique of representing musical structure in terms of its inherent patterns. It is intended to serve as a platform for further research into the perceptibility of musical structure, and also to serve as a compositional environment and ultimately as a performable instrument.

The system may be downloaded from [17] by Anonymous FTP. It includes the score file format with its interface into CM, the analysis engine, and the graphical and gestural user interface.

References

1. Howard, D.M., Angus, J.: Acoustics and Psychoacoustics. 2nd edn. Music Technology Series. Focal Press (2001)
2. Shepard, R.: 3. [24] 21–35
3. Levitin, D.J.: 17. [24] 209–227

4. Narmour, E.: The Analysis and Cognition of Basic Melodic Structures: the implication-realisation model. University of Chicago Press (1990)
5. Narmour, E.: The Analysis and Cognition of Melodic Complexity: the implication-realisation model. University of Chicago Press (1992)
6. Lerdahl, F., Jackendoff, R.: Toward a formal theory of tonal music. Jour. of Music Theory **21** (1977) 111–171
7. Lerdahl, F., Jackendoff, R.: A Generative Theory of Tonal Music. MIT Press, Cambridge, MA (1983)
8. Lerdahl, F., Jackendoff, R.: An overview of hierarchical structure and cognition. In: Machine Models of Music. MIT Press (1983) 289–312
9. Rolland, P.Y.: Discovering patterns in musical sequences. Jour. of New Music Research **28** (1999)
10. Dannenberg, R.B.: Music representation issues, techniques, and systems. Computer Music Jour. **17** (1993) 20–30
11. Bent, I.: Analysis. The new Grove handbooks in music series. Basingstoke: Macmillan (1987) with a glossary by William Drabkin.
12. Simon, H.A., Sumner, R.K.: Pattern in music. In Kleinmuntz, B., ed.: Formal Representation of Human Judgment, New York, Wiley (1968) 219–250
13. Jensen, K., Murphy, D.: Segmenting melodies into notes. In Olsen, S., ed.: Proc. 10th Danish Conf. on Pattern Recognition and Image Analysis, Copenhagen, Denmark, DIKU, HCØ tryk (2001) 115–119 Tech. Report 2001/04.
14. Bregman, A.S.: Auditory Scene Analysis: The Perceptual Organization of Sound. MIT Press, Cambridge, Masschusetts (1990)
15. Shepard, R.: 10. [24] 187–194
16. Chaitin, G.J.: On the length of programs for computing finite binary sequences. Jour. Assoc. Comput. Mach. **13** (1966) 547–569
17. Murphy, D.: Pattern Play. Anonymous FTP (2002) `ftp://ftp.diku.dk/diku/users/declan/pattern-play/`.
18. Taube, H.: Common music: A compositional language in Common Lisp and CLOS. In Wells, T., Butler, D., eds.: Proc. Int. Computer Music Conf., Ohio, USA, International Computer Music Association (1989) 316–319 Anonymous FTP from `ftp://ftp-ccrma.stanford.edu/pub/Lisp/cm/`.
19. Desain, P., Honing, H.: Quantisation routines. In Balaban, M., Ebcioglu, K., Laske, O., eds.: Understanding Music with AI: Perspectives on Music Cognition. MIT Press, Cambridge, Masschusetts (1992) 448–463 `http://www.nici.kun.nl/mmm/code/quantizers.lisp`.
20. Cemgil, T., Kappen, B., Desain, P., Honing, H.: On tempo tracking: Tempogram representation and kalman filtering. Jour. New Music Research **29** (2001) 259–273
21. Murphy, D.: Building a hand posture recognition system from multiple video images: A bottom-up approach. Technical report, cART Lab, CNR, Pisa, Italy (2002) `ftp://ftp.diku.dk/diku/users/declan/hand/cart.pdf`.
22. Bresin, R.: Articulation rules for automated music performance. In: Proc. Int. Computer Music Conf., San Francisco, USA, ICMA (2001) 294–297 `http://www.speech.kth.se/music/performance/performance_rules.html`.
23. Camurri, A., Hashimoto, S., Ricchetti, M., Trocca, R.: Eyesweb – towards gesture and affect recognition in dance/music interactive systems. Computer Music Jour. **24** (2000) 57–69 `www.musart.dist.unige.it/site_inglese/research/r_current/eyesweb.html`.
24. Cook, P.R., ed.: Music, Cognition, and Computerized Sound: An Introduction to PsychoAcoustics. MIT press (1999)