AN EXPERT SYSTEM FOR HARMONIZING FOUR-PART CHORALES¹

Kemal Ebcioglu

IBM, Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598, U.S.A.

Abstract

We describe a rule-based expert system called CHORAL, that attempts to harmonize four-part chorales in the style of J.S. Bach. The system contains about 300 rules, expressed in first order predicate calculus, and harmonizes chorales using a generate-and-test method with intelligent backtracking. A substantial number of heuristics are used for biasing the search toward musical solutions. Encouraging results have been obtained.

Introduction

In this paper, we will report on a rule-based expert system called CHORAL, for harmonization and Schenkerian analysis of chorales in the style of Johann Sebastian Bach. We will first briefly compare our approach with some current trends in algorithmic composition and music analysis, and we then will describe the CHORAL system itself.

Overview of current approaches to algorithmic music

Quite a few trends in algorithmic composition today are based on a streamlined formalism, for example, in the form of random generation of note attributes using elegant statistical distributions [Xenakis 71], terse and powerful formal grammars [Jones 81], elegant mathematical models [Kendall 81, Vaggione 84], or generalizations of serial composition procedures [Laske 81]. The economy and elegance of the formal representation underlying these musical styles (which are not in the least less respectable than traditional styles of music), may often have an aesthetic appeal in and of themselves. On the other hand, traditional music, and most of modern music, which are usually composed without a computer, do not seem to permit such economical representations. In the traditional style, the typical basic training the composer has to go through in harmony, strict counterpoint, fugue, and orchestration, already imposes a certain minimal complexity on the amount of knowledge required to describe such a style. Also, many will agree that a similar complexity can be observed in the works of modern "non-computer" composers like Karlheinz Stockhausen, Pierre Boulez, Györgi Ligeti, Jan Rychlik, and Steve Reich (his later compositions). It seems that musical composition is a hard mental task that requires a substantial amount of knowledge, and any serious attempt to simulate "non-computer" music composition on the computer would have to face the task of constructing a formal model of considerable complexity. In fact, we have found that even the algorithmic representation of the knowledge underlying the seemingly simple Bach chorale style is a task that already borders the intractable.²

As for the music analysis field, the prevailing trends seem to emphasize selective and unobvious properties of music; for example, a golden section in some motets of Dufay [Sandreski 81], or a surprising log-normal distribution in the dissonances within some chorales of Bach [Knopoff and Hutchinson 81]. Even the analysis approaches that capture a profound structure in tonal music, such as reduction techniques [Schenker 79, Lerdahl and Jackendoff 83], are still based on finding selective properties. These properties, although interesting in their own right, do not constitute a satisfactory explanation of the music in question, in the sense that there exist many "pieces" that have these properties but that have no relationship with the style. An alternative approach that would perhaps provide a more satisfactory understanding of the music, is to attempt to algorithmically generate pieces in the same approximate style. There have already been some attempts at this analysis by synthesis approach to music in some simpler traditional styles, such as folk melodies [Zaripov 60], or the first

two phrases of Bach chorale melodies [Baroni and Jacoboni 76]. But there are two problems associated with extending this approach to more substantial traditional styles: First, it may be difficult to prevent the designer of such a re-synthesis algorithm from introducing traits that would distort the style in an unscholarly fashion (this unscholarliness may be trivially removed by resynthesizing only the original pieces of the composer - but we feel that the approach of synthesizing new pieces is also interesting). The second, and more fundamental problem is that, although there has been good progress in the automated synthesis of sound [Mathews et al. 69, Roads and Strawn 85], the automated composition of non-trivial tonal music is to date not sufficiently understood (and is still regarded with some suspicion in the traditional circles). The present research is an attempt to further our understanding of the mechanical generation of music, by extending the analysis by synthesis approach to a more complex style, the style of the Bach chorale harmonizations. To cope with the complexity of the problem, we have developed a rule-based approach inspired from recent research in artificial intelligence, which we will describe below.

BSL: an efficient logic programming language

Perhaps because of its inherent difficulty, the generation of non-trivial tonal music appears to require large computational resources. In typical computing environments, A.I. languages such as Prolog or Lisp tend to be too slow for implementing our particular approach to the algorithmic generation of music. At the outset of our research, we decided to represent musical knowledge using first-order predicate calculus, and we have designed BSL [Ebcioglu 86a, 86b], an efficient logic programming language which is fundamentally different from Prolog. From the execution viewpoint, BSL is a non-deterministic Algol-class language where variables cannot be assigned more than once except in controlled contexts. However, BSL has a desirable relationship with first-order predicate calculus that makes it a new kind of logic programming language.³ A formal and rigorous description of BSL, in a style inspired from [de Bakker 79], and a proof of BSL's soundness can be found in [Ebcioglu 86a]. To implement BSL on real computers, we wrote a compiler, in Lisp, that translates BSL programs into efficient backtracking programs in C. On the simple integer computations for which it is intended, BSL tends to be about 4-5 times faster than compiled Lisp on traditional architectures like the IBM 3090 or the VAX 11/780, assuming that all available optimizations are applied to the Lisp program, such as fixed arithmetic.

A knowledge representation technique for music

To represent substantial amounts of musical knowledge in logic, it appears necessary to divide up the assertions into groups that observe the music from multiple viewpoints. The idea of using a small, nonredundant set of primitive functions and predicates to represent the music, although mathematically appealing, does not seem to be suitable for expressing all the viewpoints in a natural way. For example, a set of primitives that observes each voice as a linear string of notes, is good for expressing voice leading rules, but is somewhat unnatural for expressing harmonic relationships between voices. Similarly, a large scale work may require several hierarchical viewpoints that may constitute successively refined plans of the piece. So we opted for a representation that uses multiple sets of logical primitives to represent the different viewpoints of the music. The need for using multiple viewpoints of the solution object has arisen in expert systems in other domains as well: For example, the Hearsay-II speech understanding system [Erman et al. 80] observed the input utterance as mutually consistent streams of syllables, words, and word sequences, and the "Constraints" system of [Sussman and Steele 80] used equivalent circuits for

¹ This research was supported by NSF grant MCS-8316665, and the major portion of it was done at the Department of Computer Science, S.U.N.Y. at Buffalo, under the direction of my advisor Prof. John Myhill.

² Note that we are not referring to four-part harmonization in the context of an elementary school exercise here. When harmonizations are to encompass style specific idioms such as bold clashes of simultaneous inessential notes, without which they would be devoid of interest, the underlying knowledge tends to become greatly complicated. Traditional studies on Bach [McLose 47, Kochlin 22] have avoided the complexity of inessential notes by proceeding by examples rather than precise rules.

³ The semantics of a BSL program F is defined via a ternary relation Ψ , such that $\Psi(F, \sigma, \sigma')$ means program F leads to final state σ' when started in initial state σ , where a state is a mapping from variable names to elements of a "computer" universe, consisting of integers, arrays, records, There is a simple mapping that translates a BSL program to a formula of a first order language, such that if a BSL program terminates in some state σ , then the corresponding first order formula is true in σ (where the truth of a formula in a given state σ is evaluated in a fixed "computer" interpretation involving integers, arrays, records, and operations on these, after replacing any free variables x in the formula by $\sigma(x)$). Thus, successful execution of a BSL program without free variables amounts to a proof of the corresponding first order sentence.

observing a given fragment of an electrical circuit from more than one viewpoint.

We will now describe one possible way of implementing multiple viewpoints of the music in BSL. In this method, which was used in the CHORAL system, each viewpoint is represented by a different data structure, typically an array of records (called the solution array of that viewpoint), which serves as a rich set of primitive pseudo functions and predicates for that view. This multiple view paradigm has the following procedural aspect: It is convenient to visualize a separate process for each viewpoint, which incrementally constructs (assigns to) its solution array, in close interaction with other processes constructing their respective solution arrays. Each process executes a sequence of "generate-and-test step"s. At the n 'th generate-and-test step of a process, an acceptable value is selected and assigned to the n 'th element of the solution array of the viewpoint, depending on the elements $0, \dots, n-1$ of the same solution array, the currently assigned elements of the solution arrays of other viewpoints, and the program input. But if such an assignment is not possible, control returns to the most recent step in the history of the processes which is estimated to be responsible for the failure.⁴ The processes, implemented as BSL predicate definitions, are arranged in a round-robin scheduling chain. With the exception of the specially designated process called the *clock* process, each process first attempts to execute zero or more generate-and-test steps until all of its inputs are exhausted, and then gets blocked, giving way to the next process in the chain. The specially designated *clock* process attempts to execute exactly one step when it is scheduled, all other processes adjust their timing to this process. In certain cases a view may be completely dependent on another, i.e. it may not introduce new choices on its own. In the case of such redundant views, it is possible to maintain several views (solutions arrays) in a single process, provided that one master view is chosen to execute the process step and comply with the paradigm. An effect of rapid shift of focus between the viewpoints can easily be achieved with this scheduling technique; similarly, in large scale works, hierarchical planning can be done by having a process that constructs lower level attributes wait until sufficient input from the process that constructs the higher level plan is available (but CHORAL works without a hierarchical plan).

The knowledge base of each viewpoint is expressed in three groups of BSL subformulas, which determine the way in which the n 'th generate-and-test step is executed: *Production rules*: These are the formal analogs of the production rules that would be found in a production system for a generate-and-test application, such as Stefik's GAI system which solves a problem in molecular genetics [Stefik 78]. The informal meaning of a production rule is "If certain conditions are true about the partial solution (elements $0, \dots, n-1$, and the already assigned attributes of element n) and external inputs, THEN a certain value can be added to the partial solution (assigned to a group of attributes of element n).". Their procedural effect is to generate the possible assignments to element n of the solution array. *Constraints*: These side-effect-free subformulas assert absolute rules about elements $0, \dots, n$ of the solution array, and external inputs. They have the procedural effect of rejecting certain assignments to element n of the solution array (this effect is also called *early pruning* in A.I. literature). *Heuristics*: These side-effect-free subformulas assert desirable properties of the solution elements $0, \dots, n$ and external inputs. They have the procedural effect of having certain assignments to element n of the solution array tried before others are. The purpose of the heuristics is to guide the search so that the solution first found is hopefully a good solution. The *worth* of each candidate assignment to solution element n which complies with the constraints is determined by summing the weights of the heuristics that it makes true. Execution then continues with the best assignment to solution element n (with ties being resolved randomly), and then, if an impasse is later encountered and the intelligent backtracking algorithm of BSL returns to this step, with the next best assignment, etc.. Heuristics are weighted by decreasing powers of two; this weighting scheme was chosen because it does not involve arbitrary numerical coefficients, and because it is known to be capable of yielding good results in music generation, based on our experience with a previous strict counterpoint program where a similar search procedure was employed [Ebcioğlu 81]. For heuristics, we use a large number of concrete properties that are desirable by musical common sense; heuristics are our attempt to model the unconscious knowledge (not to be found in textbooks) that a composer uses for choosing among the possible continuations of a partial composition.

The viewpoints of the CHORAL system

We are now in a position to discuss the knowledge models, or viewpoints of the CHORAL system. We will give here only a brief overview of the knowledge base of CHORAL, which is in reality rather long and complex. The CHORAL

system uses the back-trackable process scheduling technique described above to implement the following viewpoints of the chorale:

The *chord skeleton* view observes the chorale as a sequence of rhythmless chords and fermatas, with some unconventional symbols underneath them, indicating key and degree within key. This is the clock process, and produces one chord per step. This is the view where we have placed, e.g., the production rules that enumerate the possible ways of modulating to a new key, constraints about the preparation and resolution of a seventh in a seventh chord, and heuristics that prefer Bachian cadences.

The *fill-in* view observes the chorale as four interacting automata that change states in lockstep, generating the actual notes of the chorale in the form of suspensions, passing tones and similar ornaments, depending on the underlying chord skeleton. This view reads the chord skeleton output. This is the view where we have placed, e.g., the production rules for enumerating the long list of possible inessential note patterns that enable the desirable bold clashes of passing tones, a constraint about not sounding the resolution of a suspension above the suspension, and a heuristic on following a suspension by another in the same voice (a Bachian cliché).

The *time-slice* view observes the chorale as a sequence of vertical time-slices of which has a duration of an eighth note, and imposes the harmonic constraints. This view is redundant with and subordinate to fill-in. We have placed, e.g., a constraint about consecutive octaves and fifths in this view.

The *melodic string* view observes the sequence of individual notes of the different voices from a purely melodic point of view. The *merged melodic string* view is the similar to the melodic string view, except that the repeated adjacent pitches are merged into a single note. These views are also redundant with, and subordinate to fill-in. These are the views where we have placed, e.g., a constraint about sevenths or ninths spanned in three notes, and a heuristic about continuing a linear progression.

The *Schenkerian analysis* view is based on our formal theory of hierarchical voice leading, inspired from [Schenker 79] and [Lerdahl and Jackendoff 83]. The core of this theory consists of a set of rewriting rules [Ebcioğlu 86a] which are used for parsing the bass and descant lines of the chorale separately (but we will not be able to go into the details of the theory here). The Schenkerian analysis view observes the chorale as the sequence of steps of two non-deterministic bottom-up parsers for the descant and bass. These read the fill-in view output. In this view we have placed, e.g., the production rules that enumerate the possible parser actions that can be done in a given state, a constraint about the agreement between the fundamental line accidentals and the key of the chorale, and a heuristic for proper recognition of shallow occurrences of a Schenkerian D-C-B-C ending pattern.

The CHORAL system presently incorporates about 300 production rules, constraints and heuristics. The knowledge base was developed from empirical observation of the Bach chorales [Terry 64], personal intuitions, and some traditional music treatises such as [Louis and Thuille 06] and [Koechlin 28].

Results and conclusions

The CHORAL system was originally running on VAX 11/750 and 780 computers at S.U.N.Y. at Buffalo, with BSL implemented in Franz Lisp; we have presently ported BSL and the CHORAL system to the IBM 3081-3090 computers at the IBM Watson research center, they are now running under CMS and VM/Lisp. On the basis of recent experiments on 70 chorale melodies, about 80% of the inputs take 3-13 minutes of 3081 cpu time to get harmonized, but the remaining chorales take up more time. The program takes as input an alphanumeric encoding of the melody, and outputs the harmonization in conventional music notation, and the hierarchical voice leading analysis in Schenkerian slur-and-notehead notation. We are presently making experiments to determine the factors that can produce variations in the style of harmonization for a given chorale melody. At the end of the paper, we give a recent output example, an harmonization of chorale no. 75 [Terry 64]. The voices are not in the proper ranges, but the program writes the harmonizations in such a way that there exists a transposition that will bring them to the proper ranges. Note that the parallel fifths at the end of the second phrase, resulting from the anticipation pattern in the soprano, are acceptable in the Bach chorale style (see, e.g., no. 383 [Terry 64]). However, the (eighth-eighth-quarter) patterns in the bass occurring in the beginnings of consecutive phrases, although also present in Bach's version, should probably be considered objectionable.

⁴ The generate-and-test method described here is based on the idea of producing the solution incrementally, and backing up where necessary. An alternative search technique in the field of algorithmic music is to repetitively generate a new random solution and test it, until an acceptable solution is found [Baroni and Jacoboni 76]; but this latter technique is difficult to use when the acceptable solutions are extremely few in comparison to the generable solutions, which we feel is a common situation in complex styles of music. Generate-and-test is a basic search technique used in expert system design [Hayes-Roth, Waterman, Lenat 83]. Studies on other relevant search techniques of artificial intelligence can be found in, e.g., [Nilsson 71, Pearl 83].

The harmonizations produced by the program are reasonably competent, but do not look like Bach, except for occasional "Bachian" patterns and cadence formulas. We were not able to find first principles (heuristics and ordering thereof) that would impose an austere Bachian texture in those frequent cases where a cliché does not appear to be applicable (e.g. like in portions of the exceptionally moving *Es ist genug*). Perhaps the direct algorithmic resynthesis of specific works by Bach or novel ways of incorporating statistical information from Bach's works in the knowledge base, could lead to the discovery of the correct heuristics or the missing viewpoints, but this would involve a longer research. The disadvantage of our method for algorithmic generation of music is that it requires considerable effort (generation of sophisticated large scale orchestral works may require many more rules in our estimate), and still does not allow us to develop a formal theory about the relationship between the concrete attributes of a piece and its musical beauty in the manner of an exact science: informal musical insight and patient research are still necessary on the part of the system designer(s) to make the present approach successful. We nevertheless believe that our results are more encouraging than previous attempts at tonal music synthesis by computer [e.g. Segre 81, Rader 74, Moorer 72], and we hope that the present work will be of some use to researchers in algorithmic composition who may be seeking alternative techniques.

References

- Baroni, M. and Jacoboni, C. "Verso una Grammatica della Melodia" Università Studi di Bologna, 1976.
- de Bakker, J. "Mathematical Theory of Program Correctness" North Holland, 1979.
- Ebcioglu, K. "Computer Counterpoint" Proceedings of the 1980 International Computer Music Conference, Computer Music Association, San Francisco, 1981.
- Ebcioglu, K. (86a) "An Expert System for Harmonization of Chorales in the Style of J.S. Bach" Ph.D. thesis, Department of Computer Science, S.U.N.Y. at Buffalo, February 1986.
- Ebcioglu, K. (86b) "An Expert System for Chorale Harmonization" Proc. AAAI 86, August 1986.
- Erman, L.D., Hayes-Roth, F., Lesser, V.R., and Reddy, D.R., "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty" Computing Surveys, Vol 12, No 2, June 1980.
- Hayes-Roth, F., Waterman, D., and Lenat, D.B. (eds.) "Building Expert Systems" Addison-Wesley 1983.
- Jones, K. "Compositional Applications of Stochastic Processes" Computer Music Journal, Vol 5, No. 2 (1981).
- Kendall, G.S. "Composing from a Geometric Model: *Five-leaf Rose*" Computer Music Journal, Vol. 5, No. 4 (1981).
- Knopoff, L. and Hutchinson, W. "Information Theory for Musical Continua" Journal of Music Theory 25.1 (1981).
- Koechlin, C. "Étude sur les Notes de Passage" Éditions Max Eschig, Paris, 1922.
- Koechlin, C. "Traité de l'Harmonie" Volumes I,II, III, Éditions Max Eschig, Paris, 1928, 1930, 1928, respectively.
- Laske, O. "Composition Theory in Koenig's Project One and Project Two" Computer Music Journal, Vol. 5, No. 4 (1981).
- Lerdahl, F. and Jackendoff, R. "A Generative Theory of Tonal Music" MIT Press, 1983.
- Louis, R. and Thuille, L. "Harmonielehre" C. Grüniger, Stuttgart, 1906.
- Mathews, M.V., Moore, F.R., Pierce, J.R., and Risset, J.C. "The Technology of Computer Music" MIT Press, 1969.
- Melfosse, A.I. "The Contrapuntal Harmonic Technique of the 18th Century" Prentice-Hall, 1947.
- Moorer, J.A. "Music and Computer Composition" Communications of the Association for Computing Machinery, Feb. 1972 (15).
- Nilsson, N.J. "Problem Solving Methods in Artificial Intelligence" McGraw-Hill, 1971.
- Pearl, J. (ed.) Special Issue on Search and Heuristics, Artificial Intelligence 21 (1983).
- Rader, G.M. "A Method for Composing Simple Traditional Music by Computer" Communications of the Association for Computing Machinery, Nov. 1974 (17).
- Roads, C. and Strawn, J. (eds.) "Foundations of Computer Music" MIT Press, 1985.
- Sandresky, M.V. "The Golden Section in Three Byzantine Motets of Dufay" Journal of Music Theory 25.2 (1981).
- Schenker, H. "Free Composition (*Der freie Satz*)" translated and edited by Ernst Oster, Longman 1979.
- Segre, A.M. "A System for the Generation of Four Voice Chorale Style Counterpoint Using Artificial Intelligence Techniques" Proceedings of the *Quarto Colloquio di Informatica Musicale*, Pisa, 1981.
- Stefik, M. "Inferring DNA Structures from Segmentation Data" Artificial Intelligence 11 (1978), 85-114.
- Sussman, G.J. and Steele, G.L. "Constraints - A Language For Expressing Almost-Hierarchical Descriptions" Artificial Intelligence 14 (1980), 1-39.
- Terry, C.S. (ed.) "The Four-voice Chorals of J.S. Bach" Oxford University Press, 1964.
- Vaggione, H. "Fractal C" Program notes, International Computer Music Conference 1984, Paris, October 1984.
- Xenakis, I. "Musique-Architecture" Casterman, 1971.
- Zaripov, R.K. "On Algorithmic Expression of Musical Compositions" Doklady, Akademiya Nauk, 132, 6, 1960.