9th International Conference on Music Perception and Cognition

Alma Mater Studiorum University of Bologna, August 22-26 2006

Improving algorithmic music composition with machine learning

Michael Chan
School of Computer Science & Engineering
University of New South Wales
Sydney, Australia
mchan@cse.unsw.edu.au

John Potter
School of Computer Science & Engineering
University of New South Wales
Sydney, Australia

Emery Schubert
School of Music & Music Education
University of New South Wales
Sydney, Australia

ABSTRACT

Algorithmic generation of musical sounding music is an interesting but challenging task, because machines do not inherently possess any form of creativity, which is necessary to create music. The Automated Composer of Style Sensitive Music II (ACSSM II) system generates music by searching for a sequence of music segments that best satisfy various constraints, including length and pitch range, harmonic backbone, and consistency with a probabilistic model of a composer's style. As with all optimization problems, our problem requires the construction of a search space; we utilize a clustering space produced by grouping together music segments having similar musical features. The output sequence is simply a path passing through these clusters. In order to produce such a sequence, we utilize a genetic algorithm. To evaluate the system, we have conducted an experiment, involving five subjects who possess at least three years of musical training, in which the overall musical quality of produced music was assessed. Our results show that the automatically generated music achieved a mean satisfaction score of 7.5/10, which is significantly higher than that given to the music produced by the earlier ACSSM system. Hence, the results suggest that ACSSM II is a better system than its predecessor and is capable of generating reasonably musical sounding music.

Keywords

Computer music, algorithmic music composition, artificial intelligence, machine learning, clustering, genetic algorithm, Markov chain, generative theory, musical intelligence.

In: M. Baroni, A. R. Addessi, R. Caterina, M. Costa (2006) Proceedings of the 9th International Conference on Music Perception & Cognition (ICMPC9), Bologna/Italy, August 22-26 2006.

INTRODUCTION

One of the main goals of artificial intelligence (AI) is to develop systems that imitate human intelligence and behavior, and its success inspires researchers to focus on making machines also imitate human creativity. One approach is to develop a system that generates quality music, with little or no supervision from the user. If such an automated system could create musical sounding music, it might then be useful in various ways; for example, human composers could seek inspiration for their own compositions. More generally, from a computer science perspective, such a system would represent a breakthrough in the application of AI technology.

Our system, called the *Automated Composer of Style-Sensitive Music II* (ACSSM II), aims to automate music composition by exploiting an input corpus of music and incorporating machine-learning techniques, including genetic algorithms (Goldberg, 1989) and Markov chains (Brémaud, 1999). It is based on an earlier system ACSSM (Chan and Potter, 2005) which it extends with more extensive AI techniques. Both of these systems have a common fundamental concept, which is to generate new-sounding, stylish music by "intelligently" rearranging segments of music found in the input corpus. One interesting aspect of

our system is that the kind of music style of interest is the overall style of a composer, and therefore the output music should not only be reasonably musical sounding, but also be reminiscent, in certain ways, of the composer's other works.

RELATED WORK

Compared to other areas in computing research, automated or algorithmic music generation has been relatively under-explored. Nonetheless, numerous systems have shown reasonable success in generating quality music. One of the most successful efforts is David Cope's work in Experiments in Musical Intelligence (EMI) (Cope, 1996; 2001). It is corpus-driven and adopts techniques of pattern matching, musical recombinancy, and augmented transition networks (Woods, 1970), a technique commonly used in natural language processing. Cope's philosophy is based on one of the first formal types of combinatorial music, the eighteenth-century *Musikalisches Wűrfelspiel*, and, as a result, EMI is built on the concept of recombining musical phrases found in existing works. Unlike most other algorithmic music generators, EMI not only aims at producing music that is musically pleasant, but also attempts to produce music that imitates the style of a composer. The EMI system referred to in this paper is mainly based on the earlier development in 1995 (Cope, 1995), because few technical details about the system have been published lately.

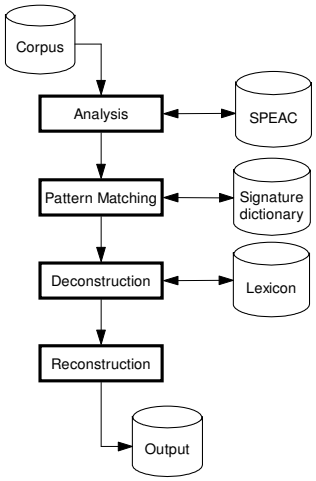

Figure 1. Block diagram of EMI algorithm.

`

As outlined in Figure 1, the structural algorithm of EMI consists of four major phases. Given a collection of works, the *analysis* phase is responsible for performing harmonic and melodic analyses based on the SPEAC identifiers, which are used to detect the "harmonic function" of a note or a group of notes. EMI exploits the observation that composers usually leave in their work some common identifiable patterns, which Cope calls *signatures.* They are collected in the phase *pattern matching*. In the next phase, the original music works are broken into small segments, or *deconstructed*, and then stored in a lexicon. Using the lexicon, these segments are retrieved and reconstructed by NLP techniques to produce the output.

Following the path of EMI, ACSSM also attempts to algorithmically produce style imitative music. Like EMI, the output music is produced by reconstructing the deconstructed music segments. Compared to EMI, the techniques used for deconstruction and reconstruction of ACSSM are improved by adopting structures proposed in a preference rule based theory called *A Generative Theory of Tonal Music* (GTTM) (Lerdahl and Jackendoff, 1983), which models the unconscious intuitions perceived by music listeners. Specifically, the lengths of segments depend on the *grouping structure* of the original music; this technique turns out to perform better than the discretely sizing technique used in EMI. ACSSM also considers the metrical aspect of music by attempting to imitate the *metrical structure* of the original music in the output music. Although the quality of music produced by ACSSM is not as satisfactory as that by EMI, the work of ACSSM nonetheless demonstrates the feasibility and possibility of improving algorithmic composition by applying cognitive and psychological techniques.

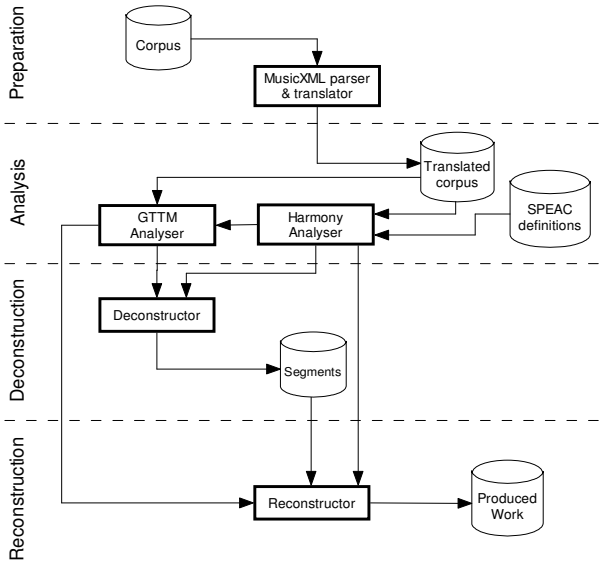

Figure 2. Block diagram of ACSSM algorithm.

The structural algorithm of ACSSM, as depicted in Figure 2, resembles that of EMI where both algorithms comprise analysis, deconstruction, and reconstruction. The corpus used in ACSSM is formatted in MusicXML (Good, 2001), a universally interchangeable language designed to model Western Classical music, and it is also widely supported by commercial music notation software, including Finale (Finale, 2006) and Sibelius (Finn and Finn, 2001). Although there are other formats available, e.g. Humdrum (Huron, 1997) and MuseData (Hewlett, 1997), parsing XML, and hence MusicXML, is a simple and efficient task. Music works stored in the corpus are first parsed and translated

ISBN 88-7395-155-4

before beginning the analysis phase, which is responsible for generating the GTTM structures.

Other music generators include *Vox Populi* (Moroni et al., 2000), which is an interactive system for music composition, based on genetic algorithms, and *Band-OUT-of-a-Box* (Thom, 2000), which is an interactive real-time improviser based on several machine learning techniques, including clustering and Markov chains.

THE COGNITION OF BASIC MUSICAL STRUCTURES

ACSSM II adopts David Temperley's *The Cognition of Basic Music Structures* (CBMS) (Temperley, 2001) to a significant extent. CBMS is based on GTTM and provides a model for musical perception and cognition with several preference rule systems. These preference rule systems cover a wide range of musical aspects, namely metrical, melodic phrasing, contrapuntal, pitch spelling, harmonic roots, and key tonics. The CBMS departs from GTTM in ways including introducing a reasonably efficient computational approach for Temperley's six models by utilizing dynamic programming techniques. With some slight modifications, CBMS structures have also been applied to Rock and African music. This paper focuses on applying CBMS to Western Classical music only, however.

Here we only review those structures of CBMS that we adapt for use in our system. In particular, we only consider melodic phrasing and contrapuntal structure.

Melodic Phrase Structure

Like the grouping structure in GTTM, the phrase structure attempts to segment a phrase into smaller sections representing the group of notes a listener tends to unconsciously perceive as connected. The phrase structure is, however, a relatively ineffective structure. Temperley argues that, owing to the complexity and ambiguity within common-place music, the melodic phrase structure is not intended, and will not suffice, to handle polyphonic music, and therefore it is designed to handle only monophonic music.

Some of the limitations of the structure are imposed by the ad hoc nature of some of the preference rules. For example, the average length of a phrase is estimated to be 8 notes, and the rule, therefore, imposes a penalty $|\log_2 N - 3|$, where N is the number of notes in the phrase.

Contrapuntal Structure

The contrapuntal analysis in CBMS is an attempt to devise a structure outlining the *melodic lines* in a music passage. It is motivated by the theory of *stream segregation* suggested by Bregman (Bregman, 1990). The contrapuntal structure focuses on the subprocess of *sequential integration*, which it is defined as "putting together events that follow one another in time". Figure 3 illustrates a contrapuntal structure of the music passage shown in Figure 4.

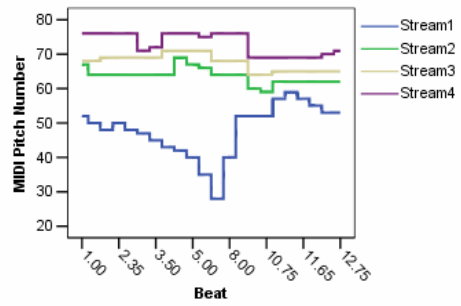

Figure 3. Contrapuntal structure of the music shown in Figure 4.

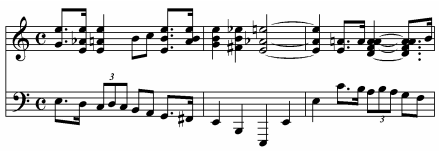

Figure 4. Measures 14-16 of Chopin's Mazuka no.50 in A min., B. 134.

ACSSM II

The intuitions underlying the extension of ACSSM into ACSSM II are threefold. First, it is desirable to allow diversity in the reconstruction process and also interesting and "surprising" elements in the reconstructed work. To this end, the system could consider not only exactly styled segments, which are what ACSSM considers, but also *similarly* styled segments. Second, the reconstructed work should be the sequence of segments that is most highly supported by quantitative evidence, for example, using probabilities so that the output is a highly probable sequence of segments that the composer might have chosen himself. Third, having seen success in applying a theoretical cognitive model in ACSSM, we can try to incorporate a computational cognitive model in our new system.

ISBN 88-7395-155-4

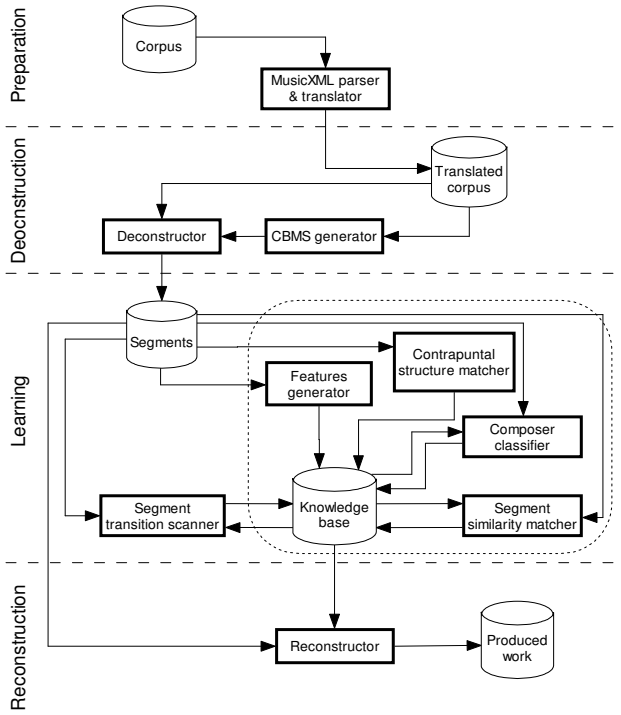

Figure 5. Block diagram of ACSSM II algorithm.

The structural algorithm of ACSSM II, shown in Figure 5, is similar to that of ACSSM. The new system inherits the use of MusicXML to format the music in the corpus and the concept of deconstruction and reconstruction. The *learning* phase is an entirely new contribution of ACSSM II. After deconstruction the learning stage applies a variety of machine-learning techniques to construct a style-sensitive knowledge base.

In the original ACSSM system the grouping structure of GTTM was used to deconstruct an input piece into segments. Our implementation tended to produce rather long segments that were sometimes still recognizable after reconstruction. Since CBMS's phrase structure is based on the grouping structure of GTTM, we used it as the basis for deconstruction in ACSSM II. As the phrase structure of CBMS can only handle monophonic music, we apply the analysis only to the highest stream of notes. Although this technique may often produce incorrect phrasing, accuracy is, in fact, not of particular importance because our goal here is merely to extract segments in a more "intelligent" manner.

After deconstruction, the learning stage concentrates on analyzing segments rather than a whole sequence. In this stage, as suggested in Figure 5, the system learns to identify *musically similar segments* (MSS) by adopting the clustering technique described in (Chan and Potter, 2006), which is depicted inside the dotted box of Figure 5. Two segments are regarded as similar only if they satisfy three criteria: they have the same composer classification according to a Naïve Bayes classifier (Mitchell, 1997); they have similar musical features in the clustering space; and they exhibit similar contrapuntal structures. A simple clustering space is shown in Figure 6, which uses two feature vectors and comprises five unique clusters. In addition to similarity recognition, a Markov chain is learnt, modeling transitions between MSSs in the style of the selected composer's music. Further details are provided in the next section.

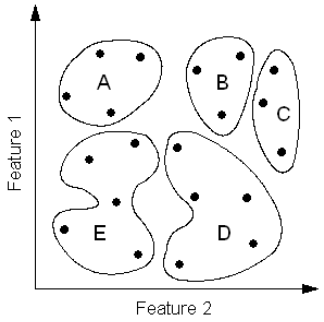

Figure 6. Clustering of segments.

Having acquired the required musical knowledge, the system then recombines segments by making use of such knowledge to *find* a good arrangement. Like our previous system, the goal in reconstruction is to recombine segments to form a new work; however, this reconstruction problem is formulated much differently in ACSSM II. The generation problem is analogous to searching for paths in a map, where the map is represented by the clustering space described in (Chan and Potter, 2006). As we will later explain, the problem can be solved by genetic algorithms.

A PROBABLISTIC MODEL OF TRANSITIONS IN A COMPOSITION

A Markov chain is used to model the probability of transitions between *generalized* MSSs. For now, we will ignore the notion of generalization. The reconstruction of the output utilizes such a model, because our intuition is that, given the sequence of MSSs of an original composition, replacing the original segments with one from the corresponding MSS should produce an equally musical sounding sequence, provided the MSSs are correctly identified. Furthermore, this technique should also be able to carry stylistic elements of the composer's style, i.e. the way MSSs are arranged, to the reconstructed piece.

With our clustering technique, most of the MSSs only contain a few segments (on average, three to six segments with a corpus of more than 26,000 segments) and so there are many MSSs. This implies that our reconstruction process, as will be explained later, may require more choices, with fewer options available for each choice. To overcome this, we relax the notion of similarity, thereby expanding each MSS which we call *generalized* MSS (GMSS). This is achieved by limiting the set of features used to assess simi-

larity. Segments belonging to the same GMSS share similar values for those features.

Most of the features have continuous values, and so our first step is to discretize them to intervals, using Fayyad and Irani's *Minimum Description Length* (MDL) discretization algorithm (Fayyad and Irani, 1993). The selection of the restricted set of features is performed via a genetic algorithm, using the Naïve Bayes classifier to provide the fitness function.

GENERATION OF STYLE-IMITATIVE MUSIC

The generation process constructs a piece by recombining selected segments into an acceptable sequence according to some quality criteria. The system attempts to find a sequence that best satisfies various measures. This reconstruction technique is reminiscent of path searching in a map with the GMSSs corresponding to cities, and transition probabilities, as well as other musical qualities, corresponding to road distance between adjacent cities.

Map search problems can be solved using a range of methods, such as depth-first or an informed strategy like A*. Uninformed searching usually takes a long time to find a good solution, partly because it has no information about the "goodness" of the current solution; informed searching, on the other hand, requires an explicit estimate of the difference in "goodness" between the current node and the solution. Because we have a large search space at hand and reasonably complex measures, making such estimates is difficult. So, we adopt a genetic algorithm approach, which can handle complex measures and generate better solutions in successive generations.

Designing the Genetic Algorithm

As genetic algorithms are capable of solving rather complex problems, they have been effectively applied to shortest-path problems, such as described in (Mowery, 2002). For path searching, it is critical to ensure the output path passes through only adjacent cities. Since genetic algorithms focuses on applying crossover between parents to produce new children, it is necessary to carefully design the crossover operator and mutation operator such that they ensure connectivity between all adjacent cities selected for solutions. It is a similar case with music reconstruction, where the crossover operator needs to maintain musical adjacency, like physical connectivity, between all consecutive segments in a sequence.

Crossover Operator

The crossover operator used for reconstruction is based on that described in (Mowery, 2002), which is designed for searching for shortest-path between two cities. In that context, the idea is to define the crossover point as the point where the two parent paths intersect; this point is, in fact, simply the location of a common city through which the two parent paths pass. This crossover method is consistent with the general principle for crossover operator, because crossing over the section before the intersection of one route with the section after the intersection of another route produces two new paths. In the music reconstruction context, such a crossover operator can be used to preserve probable musical transitions by operating on the clustering space described earlier.

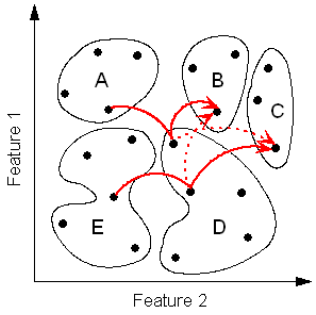

Figure 7. Example of applying the cross operator to two sequences.

Figure 7 depicts two parent sequences and their potential offspring sequences produced by applying the crossover operator described. The parent sequences and the potential offspring sequences are represented by solid and dashed arrows respectively. One parent sequence contains segments in clusters ADB and the other contains segments in clusters EDC. These two parents have a common node, as they both have a segment assigned to cluster D; therefore, the crossover operator can regard this node as the crossover point. By applying the operator, one offspring sequence will contain segments in clusters A, D and C, and the other will contain segments in clusters E, D and B. In our implementation, the selection of which segment should represent the MSS is random because segments belonging to the same MSS should be reasonably similar.

Mutation Operator

In order to prevent premature convergence, the GA implemented adopts a mutation operator that also ensures original transitivity/connectivity between the adjacent nodes in a sequence after mutation. The mutation operator, as described in (Mowery, 2002), has five functions:

- Prepend a new segment to a sequence
- Append a new segment to a sequence
- Remove a front segment from a sequence
- Remove an end segment from a sequence
- Replace the sequence with a randomly generated, but connected, sequence

These mutation changes ensure connectivity between adjacent segments, because the segment chosen for both prepending and appending must be one that belongs to a connected cluster. The last of these functions may produce

ISBN 88-7395-155-4

major mutations, but typically these will not survive into the next generation; but those that do survive may provide more discordant transitions in the musical flow.

Survival of the Fittest

Two heuristic measures are used to assess the overall quality of a candidate sequence. The goal is to determine whether a sequence is acceptable in terms of musical perception as though it is judged by a human listener.

Transition Probability

To better ensure musicality and style, we assess the quality of a sequence by examining how well the MSSs are connected. This can be done by comparing the MSSs of the segments in the sequence with the original transitions found in the corpus. To this end, the probabilistic model described earlier can be used to determine whether the probability of transitioning between two given MSSs meets a certain threshold.

Transpositional Displacement

Segments can be transposed, and we consider it advisable to avoid large intervallic displacement produced by transposition. We reward sequences with intervallic displacements equal to or less than a defined number (four, in our implementation) of semitones which limits transpositions to a major third at most.

Fitness Function

Combining the heuristic measures described above, the default raw fitness function used is:

$$fitness(c) = k \times t(c) + m \times d(c)$$

where c is the candidate sequence (the *chromosome* in genetic algorithm terminology) at hand, k and m are constant weights, $t(c)$ is the transition fitness of c, and $d(c)$ is the transpositional displacement fitness of c.

EVALUATION

We conducted an experiment, inviting subjects with at least three years of musical training, to evaluate the music produced by ACSSM II and that by the previous system. Five subjects were asked to listen to three pieces by ACSSM, and three by ACSSM II, and give a subjective appreciation score in the range 0 to 10. Such a subjective experiment allowed the subjects to decide which musical elements contribute to their musical appreciation, and so our experiment implicitly assessed various musical elements, including melody, rhythm, and harmony.

As shown in Table 1, ACSSM II produced a mean score of 7.34 and ACSSM a mean of 5.66. Table 2 shows the results from a paired sample T test of scores given to three pieces by ACSSM and by ACSSM II. Clearly, the pieces by ACSSM II are rated as being significantly more appreciable than those by ACSSM.

Table 1. Summary statistics of scores.

System	N	Min.	Max.	Mean	SD
ACSSM	5	4.20	6.40	5.66	.71
ACSSM II	5	6.70	8.00	7.46	.50

Table 2. Paired samples T test scores.

Mean	SD	Std. Err.	t	Sig. (2-tailed)
2.08	.88	.39	5.276	.006

CONCLUSION

We have designed, implemented and evaluated a new approach utilizing machine learning techniques for algorithmic generation of polyphonic music. The resulting system, ACSSM II, produces significantly more musical music than its predecessor, ACSSM. The machine learning techniques that we adopted have been applied successfully in data mining and other learning problems. We have shown that these techniques are also capable of automating music composition given the right design. Designing genetic algorithms can be difficult, due to their complex nature. Nonetheless, the operators described enable the generation of reasonably pleasant music by preserving stylistic features and basic musical elements.

Our work further demonstrates the possibility of recombining similar sounding music segments to produce acceptable, yet original, music. Furthermore, we have shown that incorporating a music cognition-based model in an algorithmic composer provides new knowledge so that it can produce better sounding music.

REFERENCES

Bregman, A. S. (1990). *Auditory Scene Analysis: The Perceptual Organization of Sound*. Cambridge, MA: MIT Press.

Brémaud, P. (1999). *Markov Chains: Gibbs fields, Monte Carlo simulation, and queues.* New York: Springer Verlag.

Chan, M. and Potter, J. (2005) Corpus-Driven Computer Music Generation Techniques. *Proc. of the 2nd Asia-Pacific Society for the Cognitive Sciences of Music*. Seoul, Korea: APSCOM.

Chan, M. and Potter, J. (2006). Recognition of Musically Similar Polyphonic Music. *Proc. of International Conference on Pattern Recognition*. Hong Kong, China: ICPR (To be published).

Cope, D. (1996). *Experiments in Musical Intelligence*. Madison, WI: A-R Editions Inc.

ISBN 88-7395-155-4

Cope, D. (2001). *Virtual Music*. Cambridge, MA: The MIT Press.

Fayyad, U. and Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. *Proc. of the 13th International Joint Conference on Artificial Intelligence*. 1022-1029. Morgan Kaufmann.

Finale 2006. [computer software] (2006). MakeMusic Coda Music Technology. *http://www.finalemusic.com/*.

Finn, B. and Finn, J. (2001). *Sibelius: The Music Notation Software*. Sibelius Software Ltd, Cambridge. http://www.sibelius-software.com/.

Goldberg, D.E. (1989). *Genetic Algorithms in Search*, Optimization and Machine Learning. Reading, MA: Addison-Wesley.

Good, M., (2001), MusicXML: An Internet-Friendly Format for Sheet Music. *IdeAlliance: XML 2001*. Florida, USA.

Hewlett, W.B. (1997). MuseData: Multipurpose representation. In E. Selfridge-Field (Ed.), *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, Cambridge, MA. 402-447. Cambridge, MA: MIT Press.

Huron, D. (1997). Humdrum and kern: Selective feature encoding. In E. Selfridge-Field (Ed.), *Beyond MIDI: The Handbook of Musical Codes*, 375-401. Cambridge, MA: MIT Press.

Lerdahl, F. and Jackendoff, R. (1983). *A Generative Theory of Tonal Music*. Cambridge, MA, MIT Press.

Moroni, A., Manzolli, J., Van Zuben, F. and Gudwin, R. (2000). Vox Populi: An interactive Evolutionary System for Algorithmic Music Composition. *Leonardo Music Journal, 10*, 49-54.

Mowery, B. (2002). Solving the generalized graph search problem with genetic algorithms. In John R. Koza (Ed.), *Genetic Algorithms and Genetic Programming at Stanford.* 158-167. Stanford: Stanford Bookstore.

Temperley, D. (2001). *The Cognition of Basic Musical Structures*. Cambridge, MA: MIT Press.

Mitchell, T. M. (1997). Machine Learning. New York: McGraw Hill.

Thom, B. (2000). BoB: An Interactive Improvisational Music Companion. *Proc. of the Fourth International Conference on Autonomous Agents*, 309-316. Barcelona, Spain: ACM Press.

Woods, W. A. (1970). Transition Network Grammars for Natural Language Analysis. *Comm. of the Association for Computing Machinery 13, No. 10*, 591-606.

ISBN 88-7395-155-4