

Automatic Music Composition with AMCTIES

Nuwan I Senaratna

University of Colombo School of Computing

Email: nis_nisco@yahoo.com

Address: 32, Temple Road, Colombo 10, Sri Lanka

Telephone: 0714-163-477

Abstract

A wide range of approaches and techniques have been explored in attempts to build Automatic music composition (AMC) systems. However, most attempts have tended to use a single, centralized technique for the entire AMC process. This has many drawbacks. This paper proposes a solution to these drawbacks in the form of a tree-like structure made up of several interacting emergent systems. The structure which will be referred to as "A Tree of Interacting Emergent Systems" or TIES, is a context-independent framework, applicable to many applications, including AMC. The latter is demonstrated by using TIES to implement the application AMCTIES - An application capable of "Automatic Music Composition using a Tree of Interacting Emergent Systems". This paper would be of much interest to those in both the computer as well as music fields.

Keywords: Automatic Music Composition, Emergent Systems, Cellular Automata, Genetic Algorithms, Fractals.

1. Introduction

Musicians and scientists alike have been very keen to understand how exactly humans compose music. Automatic Music Composition (AMC) has, hence, quite naturally, become a field of much interest. Many scientists have pursued the Subject in an effort to come up with a human-free, real-time music composing system. Also, due to its obvious commercial potential, industrialists in the cellular phone and video game industry have been keenly monitoring developments in this area [20].

Many computational techniques have been used for AMC. Conventional methods have been tried, but like many applications that attempt to mimic human behaviour, these have not been very effective [20].

Of late, there have been many attempts at AMC using fresh AI related techniques. Of these "Emergent Comput-

ing" techniques have shown much potential. This is particularly because it is possible to draw close analogies between "emergence" and human composing technique. However, attempts so far have been marred by several problems. This is chiefly because most Emergent AMC systems tend to use a single technique - a lone, centralized emergent system that attempts generating all the material required.

The approach presented in this paper, attempts to remedy these problems by replacing the traditional single technique approach with TIES - A Tree of Interacting Emergent Systems. TIES is a tree-like macro-structure made up of several interacting emergent systems. It is a context-independent framework and is applicable to many applications, including AMC.

2. Background

2.1. General drawbacks of AMC attempts

Lack of domain knowledge: The chief problem with attempts at AMC so far, has been the lack of incorporation of domain specific knowledge on the part of the application designers. In this case the "domain" is music and its composition process. Many of the AMC techniques that have been recognized as "good" are often very domain specific. Using domain knowledge in implementing systems is vital [10].

Technique specific applications: Many attempts at AMC seem to have concentrated on the technique used rather than the problem at hand. For example, many neural network based efforts seem to concentrate on improving existing generic neural network designs rather than devising better AMC schemes [13]. Although such attempts at technique enhancement have resulted in interesting and useful findings, they are not helpful in the context of this AMC.

Separation of Music from Computing: As a result of the previously mentioned "technique specific-ness", the na-

ture of the musical output is tightly coupled to the computer generation technique that produces it. Ideally, this should not be the case. Suppose a user makes a request to the AMC systems to produce some musical element (for example a Rhythm). The AMC system should find an appropriate generator (out of possibly several generators) that could service the request, and then ask it to so service it.

This would also make it possible for a more “distributed” implementation of AMC; rather like a group of composers cooperating to produce a piece of music. All the composers need not be available at a given instance. Nor does the user need to know about their existence.

Imitating and not creating: Many AMC applications seem to be only good at boring, imitations of existing compositions. Many simply provide variations on existing works, rather than generating new and creative works. This problem can be attributed to the over reliance on rule-based strategies and too conventional implementation techniques [3].

Boring and predictable results: Another bad result of over reliance on rule-based strategies and too conventional implementation techniques is that the resultant output is highly predictable. Unpredictability is an essential part of creativity.

Lack of control over the nature of the output: At the other extreme of things, techniques using “pure” emergent techniques (for example pure fractal based melodic composition or purely cellular automata based music composition [2],[6]), the resultant musical output tends to be ad hoc and meaningless. The user (or initiator) of the process has little control over the results. In practical situations this is a serious problem; what is the use of an AMC system if it cannot produce what the user wants? [15]

Lack of Flexibility: As mentioned, most attempts at AMC use a single method. Though some applications have fairly elaborate musical options, few allow the flexibility to change the generation technique. It is reasonable to expect new and better techniques of generation coming into being as time goes on. A good AMC system must be flexible enough to incorporate these.

Lack of Scalability: An extension of the lack of flexibility is the lack of scalability. Musical compositions vary considerably in scale and variety. Some compositions might be very simple, while others are complex and elaborate. A good AMC system must be scalable. It must be minimally possible to combine two systems to form a composite system. Current attempts at AMC do not allow this.

2.2. Reasons for selecting Emergent Techniques for AMC

Modeling the human compositional process: As explained, a musical composition, as a whole, consists of many characteristic elements. These characteristic elements interact in complex and not-always-obvious ways. By themselves, the elements are often very short and trivial, with no apparent musical meaning. It could be said that, when a human composes a piece of music, what he actually does is to arrange these characteristic elements such that his aesthetic and expressive goals are achieved [1]. Hence, at least on the surface, the human composition process itself seems to be an emergent system. Therefore, it is logical that emergent techniques are used to artificially model it.

Nature offers many examples: Someone might argue that the previously expressed connection between the human music composition process and emergent techniques is only superficial (or even non-existent). Such an argument is, in a way, not unreasonable as the human brain has not been understood to a degree to logically justify such a claim. However, we have much indirect evidence to justify the use of emergence. Much of this comes from nature. For example, the branching of blood vessels in our circulatory system, the emergence of tree rings on the barks of trees and even weather patterns have been shown to follow the behaviour of emergent systems [19].

Flexibility: As explained, there are many different ways in which AMC can be approached, and many different ways in which each approach can be implemented. In spite of drawbacks, each approach has good features as well. An advantage of using emergent techniques is that they can be adapted to incorporate good features of other techniques. For example, it is possible to incorporate statistical (especially stochastic) features into Cellular Automata [7] and fractals. Genetic Algorithm fitness functions can easily encode Rule-based knowledge and Grammars [5], hence, exploit their advantages, while at the same time avoiding disadvantages[9], [11], [12].

Robustness and Speed: Emergent systems have no single-point of failure; if a single unit fails, the system still works. Emergent techniques tend to find a reasonable solution quickly and then optimise. Traditional computer algorithms tend not to produce a useful result until they are complete.

Interplay between local randomness and global determinism: Creativity can be “generated” in two ways in a computer. Firstly, the generation of random musical entities can be considered “creative”; after all, random data

can be said to have “never been seen before”. Secondly, if these random entities were composited to form some sort of higher level entities, these higher level entities could also be “creative”. Note that in the second case the results might cease to “appear” to be random [19].

Generally these two kinds of creativity can be referred to as creativity resulting from local randomness and creativity resulting from global determinism. Emergent systems combine both these techniques in varying degrees, and hence allow a system capable of producing maximal “creativity”.

Stable Structure and error tolerance: A disadvantage of using conventional, non-emergent techniques is that these do not allow flexibility in terms of the material generated. Any attempt at flexibility is resisted or else results in too many errors. Conversely, emergent systems have some degree of error tolerance and as a result give the system some extra stability.

3. The Design

3.1. Overall Architecture

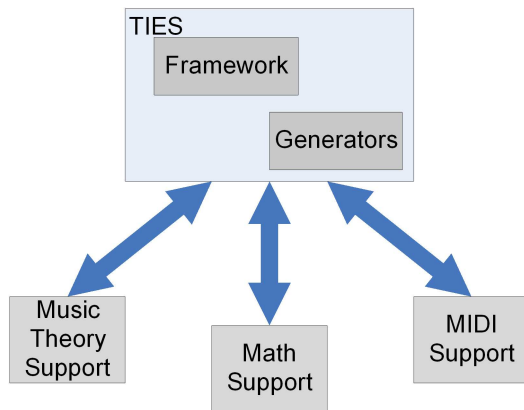


Figure 1. Overall Architecture (Design Diagram)

At the core of the AMCTIES architecture, is a tree of interacting emergent systems (or TIES for short). TIES consists of a framework for implementing emergent systems and the implemented emergent system themselves (which we will refer to as generators).

In addition to this, TIES needs several support modules. Since AMCTIES is a music composing application, music related constructs and algorithms need to be defined and implemented. Also, since the output of AMCTIES is for the MIDI format, we need support for writing musical compositions to MIDI files as well. Finally, we all these

modules implement several mathematical functions. Support for these must also be provided.

Hence, the overall AMCTIES architecture consists the “framework” and the “generators” in TIES and support modules for Music Theory, MIDI and Mathematics.

3.2. TIES (A Tree of Interacting Emergent Systems)

Generically, TIES (a Tree of Interacting Emergent Systems) consists of a cascading tree of emergent systems. The systemic properties of the root node emergent system are the actual output of the system. The low-level entities of each (non-leaf) node emergent system consist of its children’s systemic properties.

For example, for this AMC application, the TIES root node will be an emergent system that produces the final piece of music. The low-level entities out of which this emerges are the systemic properties of child emergent systems that produce Subjects. These, in turn, have child emergent systems that produce Phrases. Finally, at the bottom of the tree are the lowest-level emergent systems that produce Motifs, Rhythms, Dynamics, Timbre and Chords. The user perceives the system as one emergent system that generates movements of music. For explanations on musical terms see [8], [16], [18].

The TIES framework, though ideal for the kind of AMC application pursued in this paper, is not limited to AMC. It might be used in other areas of computer science as well.

TIES is, as mentioned, itself an emergent system. However, it has several advantages over monolithic emergent systems:

Context Sensitivity: A general problem with all techniques used in AMC applications (including emergent systems) was the lack of context sensitivity. Although a given technique might be good for implementing certain parts of an application, it is generally not suitable for implementing all application aspects. As seen, many AMC applications use a single technique to accomplish “too many” tasks. It is more the pity since many techniques are actually quite good for specific tasks.

TIES solves this problem by having different systems with multiple capabilities hidden inside its structure.

Controllability: Most complex systems (including emergent systems) are difficult to control. For example, in a conventional emergent system, it is difficult to control how exactly low-level entities influence high-level systemic properties.

In TIES, it is possible to control interactions between the component node emergent systems (even though it might not be, in general, possible to control interactions within them). This gives better controllability, especially of design

parameters that, often, directly influence the efficiency of the application.

Higher irreducibility: The fact that TIES is more controllable does not in any way make it more reducible. TIES is a function built upon the nesting of several irreducible functions. Since, the interactions within it do not, in general, force trivialization, TIES itself is highly irreducible.

The concept of “irreducibility” in emergent and other complex systems has been coupled with the concept of “non-linearity” (the fact that small changes can lead to big effects). The fact that it is made up of a hierarchy of inter-connecting systems that each display non-linearity, multiplies the effect of non-linearity in TIES itself.

Higher unpredictability: As was the case with irreducibility, TIES is also relatively more unpredictable. This is particularly important in the case of AMC since there is a very real need for creativity and originality.

Flexibility: All that is required of each individual node in TIES is to play some specific role. Since, it is the role that is important, the actual implementation might be changed. TIES serves as a flexible framework that defines interactions, not specific implementations. Hence, it is possible to change actual implementations at will, when required.

Scalability: A direct implication of flexibility is scalability. Although this does not seem directly relevant to AMC systems, future applications might benefit from this.

Parallel Implementability: Since TIES has a distributed node-based structure, it has a high potential for parallelization. This might be important for real-time AMC applications.

Reusability: As mentioned, TIES is a collection of nodes that play several different specific roles. These roles are, in general, specific parts of an application whole. The framework of TIES allows nodes to be implemented in independent modules, allowing free code reuse.

3.3. Design Components

In applying TIES to AMC, the “emergent systems” are a set of algorithms capable of generating musical entities (which are the “emergent objects”). Emergent entities belong to a certain Emergent Object Type. Specifically, the emergent object types in AMCTIES are Motifs, Rhythms, Chords, Dynamics, Timbres, Phrases, Subjects and Movements. The emergent systems will be Motif Generators, Rhythm Generators and so on.

Generators and emergent entities are the two fundamental design components of TIES. In trivial cases, these two alone will be sufficient to implement TIES; however, this is not so for more complex ones.

To see this let us first consider a simple case where we have a single Motif Generator, a single Rhythm Generator and so on. Also suppose that generating high-level emergent objects requires exactly one of each of its immediate low-level objects. For example, suppose the Phrase generator needs exactly one Motif, exactly one Rhythm and so on. This can be very easily implemented. We generate a Motif using the Motif Generator and store it in some temporary variable. We do the same for the Rhythm, Chords etc. Finally, we retrieve the stored low-level objects and pass it to the Phrase Generator to generate the Phrase. The process is continued to generate the Subject and the Movement.

Note: From here on when speaking in the context of TIES, by “generating high-level emergent objects from low-level objects” we mean “generating high-level emergent objects from the low-level objects in the immediate lower level”.

3.3.1 Emergent Object Collections

In most cases several low-level objects would go into generating a single high-level object. For example, we need several Motifs to generate a single Phrase. To complicate matters further, several low-level objects might be used to generate several high-level objects. For example, several Phrases might be based on several Motifs. However, not all low-level objects might be suitable for generating a particular high-level object.

This poses two design requirements: Firstly, we need some mechanism of temporarily storing low-level objects. Secondly, we need to specify what low-level objects should be used to generate each high-level object.

We satisfy the first requirement by implementing design components known as Emergent Object Collections. An Emergent Object Collection is simply an array storing pre-generated emergent objects. The objects might be accessed via the array index. For each emergent entity type, we have exactly one Emergent Object Collection. For example, a TIES implementation will have exactly one Motif Collection, exactly one Rhythm Collection and so on.

3.3.2 Maps and Map Elements

The second requirement is satisfied by implementing design components known as maps. A map simply “maps” low-level objects to high-level objects. For each high-level object that need be generated, a map specifies what low-level objects should go into the generation process. The low-level object is specified via its array index in its respective collection.

Since a high-level object usually corresponds to several low-level objects, the indices of these low-level objects are stored in as a single structure known as a Map Element. For example, a Phrase Map Element will consist of an index of an element in a Motif Collection, an index of an element in a Rhythm Collection and so on.

3.3.3 Generator Collections

To produce a variety of musical styles we would have to use several generators for each type of emergent object. For example, we might have a Fractal Chord Generator and a Cellular Automata Chord Generator. For this reason, we need to also specify what generators should be used in generating each high-level object. This too can (and is) specified in the map. We can store all the generators in an array, which we will call a Generator Collection and access the element generators via its array index. Hence, (finally) a map contains an index to a generator array and a set of map elements.

3.3.4 Map Collections

We group all the map elements for each emergent entity type in a single array which we will refer to as a Map Collection. For each emergent entity type, we have exactly one Map Collection. For symmetry, we can implement the Map Collection such that a map pertaining to a particular object is indexed by the objects index in its respective Emergent Object Collection.

3.3.5 Lowest level emergent objects

Above, we have partially assumed that all emergent objects are generated using lower-level emergent objects. This is not true for the lowest-level emergent objects (for example, Motifs, Rhythms etc.). In fact in the case of AMCTIES most of the emergent object types are at the lowest level of the tree. The design components have to be slightly modified to accommodate these. Lowest-level emergent objects have no map elements. Corresponding maps will only specify the generator index. Hence, such maps might be specified as an integer array.

3.3.6 Map Generators

Maps are generated via design components known as Map Generators.

3.4. Designing Generators

3.4.1 Motif Generation

A Motif is simply a sequence of pitches. The only constraints that must be adhered to when generating Motifs is

that certain pitch relationships between pitches must be respected [16].

Random Fractals: Random Motif Generation can be implemented in two ways. First, we could generate a random sequence of pitches, and then modify certain notes that may not respect any undesirable pitch relationships. There are two types of such undesirable relationships: One, pitches that do not correspond to the scale of the Motif. Two, notes that have either too high or too low pitches. These can be filtered out by suitable methods.

Second, we could generate a sequence of pitches at random, but from a set of pitches that are guaranteed of satisfying the above pitch relationships.

Random Motif Generation is very simple to implement and is very efficient. The randomness gives a natural creativity and emergence. It could be argued that this method is “ad hoc” and no rational pattern could emerge out of this. This argument might be countered with two points: One, Motifs are very short entities and it is difficult to spot rational patterns in them. Two, qualities that might emerge out of a process of generating Motifs have less to do with structure and more to do with melody. However, this view might be considered subjective; hence, we will look at some other methods of generating Motifs as well [4].

Iterated Function System Fractal: This will exploit the fact that within Motifs certain symmetries exist. This fractal implementation is intuitive as a pitch is dependent on its predecessor and successor. The generation function can be implemented in several ways: We can use a deterministic table which maps (Predecessor, Successor) pairs to generator point values. We can also make this same process stochastic by making the table probabilistic (For example, use a second order Markov Chain). The building of the table can be done either according to known musical relationships or using some other process (We could even incorporate this into TIES by using an emergent technique. If the initial state is the same and the number of generations is constant, the deterministic technique will always return the same result. Hence, it is vital (for variety) that these variables (initial state and number of generations) are changed [4].

Escape Time Fractal: Compared to the IFS technique, using Escape Time Fractal will give more variety since generation depends on only one factor (the Predecessor). As before, the generation function can be either deterministic or stochastic (The same design decisions discussed also apply). Additionally, however, to prevent the fractal from growing out of range, it must be bounded [4].

There are two implementation problems in Escape time fractals. Firstly, they might not be suitable for relatively short Motifs. This is particularly problematic as we will be

implementing Motifs as lowest-level entities in TIES. Secondly, the functionality of Escape Time Fractals (at least in this context) can be mimicked to an extent by Random Fractals.

3.4.2 Rhythm Generation

Random Fractals: As with Motifs, the random technique has also been used for Rhythms. The technique used here is as follows. Initially, the Rhythm consists of a single pulse with duration equal to the duration of the entire Rhythm. The pulses in the Rhythm are iteratively selected and split randomly [4].

Iterated Function System Fractal: An Iterated Function System based technique has been used to generate Rhythms. As in the case of Motif note sequences, it is essential that Rhythms form some coherent structural shape [4].

Beat Rhythm Generation: In many musical works, Rhythms follow standard well-known patterns. In such cases, the Rhythms can be hard-coded, and it is not necessary to use fractals or a similar generation method. A single piece of music might have a well-known Rhythm pattern intermingled with more novel patterns; hence, it is important that both strategies of Rhythm generation co-exist [4].

3.4.3 Timbre Generation

Fixed Timbre Groups: Usually, the Timbre of music is fairly fixed. In cases where there are more than one instrument playing simultaneously, these are assigned specific parts. Hence, we will not use emergence for generating sequences of Timbre possibilities. We will simply assign constant instruments for each MIDI channel. For details on MIDI see [17].

3.4.4 Dynamic Generation

Fractal based Dynamic Generation: An Iterated Function System based technique is used to generate a Dynamic pattern. A Dynamic pattern in a piece of music is a sort of time series. On the other hand, it can also be considered a sort of architectural shape, with peaks and valleys. Iterated Function System Fractals are ideal for generating this behaviour. However, since AMCTIES is essentially a music composing application (and not a sound generating one), it might be argued that elaborate dynamic directions are superfluous. We might be better off with a simpler design and leave the dynamics to the discretion of the performer.

Constant Dynamic Generation: Unlike Motifs, Rhythms, Chords and Timbres, in works of music, the Dynamic variety is usually left to the discretion of the performer. Hence, it is not vital that the exact Dynamic directions are specified. Also, modifying the Dynamics of a piece would not (usually) change its style or character that much. Hence assigning a constant value for Dynamics would not impair the compositional process too much, and would also add to practicality and simplicity.

3.4.5 Chord Generation

Fractal based Chord Generation: An Iterated Function System based fractal technique provides a very intuitive strategy for generating sequences of Chords. A Chord is dependent on its predecessor and successor. Conversely, given a predecessor and successor it is possible to deterministically (or probabilistically) say what the Chord should be. This technique is very similar to the IFS fractal based technique for generating notes.

However, unlike in the case for generating note sequences, the nature of Chords is governed by harmony. Hence, in this case, we are constrained by harmonic rules. So we have less freedom in specifying deterministic (or probabilistic) rules. This results in reduced creativity.

Conversely, we have the option of ignoring music theory constraints in order to give emergence a freer hand. However, this could result in some “unmusical” sounding output [14].

Cellular Automata based Chord Generation: The principal behind using Cellular Automata for Chord generation is the same as that in using fractals. We use the fact that a Chord is dependent on its predecessors and successors.

However, in the case of Cellular Automata we can explore these dependencies more deeply. For example, we can use an evolution function that considers a neighbourhood that goes beyond immediate predecessors and successors. We can also consider relationships between individual notes within Chords, by using a two dimensional Cellular Automata [7].

As in the case of fractal based generation, we are contained by music theory rules.

Standard Chord Generation: As in the case of Rhythm, many “standard” Chord progressions are found in pieces of music. Certain kinds of pieces involve the repetitions of the same Chord sequence over and over again. Hence, it is important that this is implemented as well [14].

3.4.6 Phrase Generation

Genetic Algorithm based Phrase Generation: The emergent entities considered thus far were lowest level entities; that is, they are not generated using other emergent entities. A Phrase is generated using a set of Motifs, Rhythms, Chords, Timbre and Dynamics. In our TIES design these have been generated earlier and stored as emergent object collections. The exact low-level entities to be used in the generation of the Phrase are defined in a PhraseMap. Hence, part of the “generation” of the Phrase is defined in the Phrase map – however, the Map generation process is fairly simple and unimportant in this context [9], [11].

Generating a Phrase does not consist of simply merging the low-level material together. It must be verified that these are compatible. For example, we must check if a certain Motif is compatible with a Chord. If not the Motif must be modified to fit the Chord. Also, Motifs, Chords, Dynamics and Timbre might not be compatible with the Rhythms. For example, the Motif might contain more (or less) pitches than the Rhythm. If this is the case, the Motif must be cropped (or extended) artificially. Finally, when several Motifs are concatenated to form the Phrase, we must make sure that the end of one Motif is compatible with the beginning of the next Motif.

The above “verification of compatibility” is not a trivial task. Genetic Algorithm based Phrase Generation consists of performing the “compatibility assuring” tasks using Genetic Algorithm techniques. For specifically the Genetic Algorithm tasks are as follows:

- Given a Rhythm and Motif, we evolve the musical material to find some coherent NoteSequence (Actually this should be “Given a Rhythm, Motif. Timbre and Dynamic”. However Dynamics and Timbre are fairly independent of fellow low-level entities and can be exempted from this process).
- Given the above evolved NoteSequence and a Chord, we evolve the NoteSequence to suitably fit the Chord.
- Given a set of NoteSequences, we evolve them so that their concatenation will form a coherent Phrase.

3.4.7 Subject Generation and Movement Generation

Fixed Subject Generation and Fixed Movement Generation: Subjects and Movements are high-level musical entities. This has some pertinent consequences. Since they are temporarily fairly long, a single piece of music has relatively few of them. For example, a movement in what is known as “Simple Rondo Form” consists of just five Subjects $A1+B+A2+C+A3$ (Here $A1$, $A2$ and $A3$ might be identical or slightly varied). For this reason, it is difficult to give creativity or novelty to the piece by controlling the

generation process. Subjects and Movements are essentially structural entities and are there to give the piece an architectural foundation, not variety or colour. Conversely, following fairly set patterns in music theory in generating Subjects and Movements would not reduce the creative nature of the piece [5].

Hence, we follow a “fixed” approach to Subject and Movement generation. That is when combining Phrases to form Subjects we will simply follow standard musical rules; we will not use any emergent techniques. The same applies to combining Subjects to form Movements.

4. Summary of Implementation

The implementation of emergent objects consists of two parts. First, the actual data is represented as some data structure. Motifs, Rhythms, Timbres and Dynamics are represented by an integer array. Chords are represented by an array of integer arrays. Finally, phrases, subjects and movements are represented by “NoteSequences”. A NoteSequence is simply a time-ordered sequence of data structures representing notes. Second, an interface specifies how the generator for that object must be defined.

Motif, Rhythm, Chord, Dynamic and Timbre are not made up of further low-level entities. Hence, maps corresponding to these simply store the indexes of their generators, and can be implemented as integer arrays. The higher emergent entities require a more complex implementation. For example, a Phrase Map Elements is a structure pointing to a Motif, Rhythm, Chord, Dynamic and Timbre object, while a Phrase Map contains a pointer to a Phrase Map Element array and a Phrase generator.

AMCTIES is a framework that contains several techniques for generating a range of different types of music. As it attempts to mimic human musical composition, certain aspects of a human’s environment need to be represented within the framework. This can be done by the user selecting what generators to use and other options. A simple class “Global” defines an abstract representation of the User’s Environment. This consists of several option variables, public method Generate and several supporting private methods. The actual user environment is represented by extending this class.

Each of the emergent object generators were implemented in a corresponding class. The implementation was by and large mapping the design to algorithms and then to code. Since most of the algorithms are well known and have been used in other applications before, for clarity these will not be discussed here. The implementation of the support modules (MIDI, Music Theory and Math) will also not be discussed.

After implementation, some of the algorithms were tuned for optimal performance. Genetic algorithm based

techniques were tuned to find the optimal population size, tournament size, number of generations, mutation and crossover rates etc [12]. The criterion used for judging the tuning was processing time and result quality. Since, Cellular Automat and Fractals were used for low-level entity generation, they did not require as much tuning.

5. Comments on Results

5.1. Summary of Expert Evaluation of Output

5.1.1 Summary of the process

The test cases consisted of a collection of varied pieces of music generated by AMCTIES. Various parameters were given to generate varied pieces (e.g. pieces in different styles). However, after generation no selection or pre-judging was done on the test cases. Hence, all the test cases were as AMCTIES produced them and free from any tampering. Also, there was no attempt to improve on the pieces by other means (for example, by editing them on a MIDI editor) after they were generated.

The test set consisted of 46 samples, each around 30 seconds to 5 minutes long. The upper limit was due to practical constraints (AMCTIES is capable of generating arbitrarily long pieces of music).

An expert session consisted of the expert validating 30 randomly selected test samples (out of the total 46). A session typically lasted around 40 minutes, with 30 minutes for listening and 10 minutes for taking down responses.

The tests consisted of a series of yes/no questions asking for a comparison between the output produced by AMCTIES and “a human musician with a fairly advance musical talent and education”. The questions were adapted from a set of guidelines given to examiners marking examination questions in music composition. Different validation criteria were used for various aspects of the musical piece. These were broken down into three areas: local structural features (Motifs, Rhythms, Chords, Dynamics and Timbre), global structural feature (Phrases, Subjects and the Movement as a whole) and stylistic features (historical relevance, expressiveness, creativity etc.) The main reason for using a simple yes/no test was that it follows along the lines of the Turing Test. It also takes some of the subjectivity out of the test. It also helped pinpoint definite weaknesses or strong points in the system, without ambiguity.

The tests were restricted to two independent “experts” who are professionals in the music field with training and professional qualifications in performance, teaching and examination.

5.1.2 Summary of the results

Motifs: The experts agreed that there was considerable variety in the range and style of Motifs. There was a range of colours and effect. The Motifs were very interesting, seldom dull.

Rhythms: As with the Motifs, there was considerable variety in Rhythm. The accentuation was regular and coherent. Some Rhythms were sometimes indeed unorthodox, but this often added to the novelty of the music.

Chords: Most of the harmony was regular and correct. However, there were certain unorthodox Chords that seem to give the impression of “wrong notes”. However, even the “wrong notes” seemed to be put there for “right reasons”, in that they seemed to form a coherent whole.

Dynamics and Timbre: The actual emergent techniques used in AMCTIES did not attempt at controlling Dynamics and Timbre, actively, since it is essentially a music composing application, not a sound generating one. The Dynamics and Timbre were essentially added artificially. However, it was essential that the other musical entities (Motifs, Rhythms, etc.) complement these. The general expert opinion was that this was the case. They also provided variety and effect.

Phrases and Subjects: The structure and arrangement of Phrases and Subjects enable satisfactory Motif development. The developmental lines were sometime blurred. However, this was recognized only on close scrutiny of the score; as far as listening was concerned, the form and structure was good.

Movements as a whole: The complete musical works produced could be compared to a human composer with a fair composing ability and in certain sections “very clever” devices were used. The music was ideal at creating atmosphere and emotion specific sounds. The flow of some of the music was very soothing. The system proved capable of generating music in several contrasting styles. The music was generally expressive and was sufficiently interesting enough to hold the listener’s attention.

5.2. Summary of Comparing output with other compositions

5.2.1 Summary of the process

The AMCTIES test set consisted of the 46 samples described earlier. The “other set” consisted of 74 samples of music which consisted of samples generated by other AMC

systems and MIDI versions by some not too well known compositions by human composers.

Random samples were picked from each set (with each set having an equal probability of being selected). A programme consisting of these samples were presented independently to experts.

5.2.2 Summary of the results

Motifs: There was no considerable difference between the output of AMCTIES and other systems (in general) in this respect. However, AMCTIES did have considerably more variety than certain AMC systems (especially those based on fractals). These other systems produced output that tended to be too repetitive, while AMCTIES reached a better balance between repetition and variety.

Rhythm: Similar observations (to Motifs) can be made here. However, again, in general AMCTIES produced more controlled output.

Chords: Most other systems completely lacked harmony. Most had totally random harmony (that is with no actually conscious attention to harmonization at all). Others had one or two Chords that repeated in a highly monotonous manner. Those that had few parallel parts sounded better than the others for this reason. AMCTIES was far superior. It had instances of both conventional and unconventional harmony. The important point in both these cases was that AMCTIES produced coherent Chords.

Timbre and Dynamics: AMCTIES was similar in this aspect to most other applications. Not many deviations were observed. Also, since Timbre and Dynamics can easily be added artificially, making comparisons on these grounds might not be valid or fair.

Phrases and Subjects: Most other applications were simple in form and structure. Conversely, AMCTIES is capable of elaborate forms and structures. As the expert evaluation revealed earlier, large forms can be achieved with coherence.

Movements as a whole: In general it is evident that AMCTIES is capable of producing a very large range of musical styles. In contrast, most other systems are restricted in what they can produce. Though these systems might produce a limited range of styles well, they are incapable of producing other styles. Hence, AMCTIES has the functionality of several other systems combined.

6. Conclusion and Future Work

AMCTIES has been evaluated by experts in the musical field. They are confident about the musical merits of its output and rate its computational ability favourably. Music produced by AMCTIES was found to be expressive, naturally structured and interesting.

AMCTIES or Automatic Music Composition using a Tree of Emergent Systems seems to have succeeded in achieving its goal as a system that mimics the human music composition process. It has several interesting features and implements many novel techniques. It can also be considered as an excellent foundation for developing bigger and better music composition applications in the future. It is likely to prove useful for professionals in both the music and the computer fields alike.

References

- [1] J. Braun and D. E. Linder. *Psychology Today*. 1975.
- [2] D. Burraston, E. Edmonds, D. Livingstone, and E. R. Miranda. Cellular Automata in MIDI based Computer Music. *Proceedings of the 2004 International Computer Music Conference*, 2004.
- [3] G. Buzzanca. A Rule-Based Expert System for Musical Style Recognition. *Proceedings of the 1st International Conference Understanding and Creating Music, UCM 2001, Caserta*, 2001.
- [4] D. Conklin. Music Generation from Statistical Models. *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences, Aberystwyth, Wales*, 2003.
- [5] A. O. de la Puente, R. S. Alfonso, and M. A. Moreno. Automatic composition of music by means of Grammatical Evolution. *APL'2002 Madrid Proceedings*, 2002.
- [6] A. Dorin. LIQUIPRISM: Generating Polyrhythms with Cellular Automata. *Proceedings of the 2002 International Conference on Auditory Display, Kyoto, Japan, July 2-5*, 2002.
- [7] N. Ganguly, B. K. Sikdar, A. Deutsch, G. Canright, and P. P. Chaudhuri. A Survey on Cellular Automata. *Technical Report Centre for High Performance Computing, Dresden University of Technology*, 2003.
- [8] A. Jacobs. *Penguin Dictionary of Music*. 1997.
- [9] B. Johanson and R. Poli. GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 1998.
- [10] H. Jrvlinen. Algorithmic Musical Composition. *Seminar on content creation, Telecommunications software and multimedia laboratory, Helsinki University of Technology*. April, 2000.
- [11] M. Marques, V. Oliveira, S. Vieira, and A. C. Rosa. Music Composition using Genetic Evolutionary Algorithms. 1998.
- [12] M. Mitchell. *An Introduction to Genetic Algorithms*. 1996.
- [13] M. C. Mozer. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multiscale process. *Connection Science*, 1994.

- [14] F. Pachet and P. Roy. Musical Harmonization with Constraints: A Survey. *Constraints*, 2001.
- [15] A. Santos, B. Arcay, J. Dorado, J. Romero, and J. Rodriguez. Evolutionary Computation Systems for Musical Composition. *Mathematics and Computers in Modern Science*, World Scientific and Engineering Society Press, 2000.
- [16] P. Scholes. *The Oxford Companion to Music*. 1998.
- [17] M. M. A. Website. www.midi.org.
- [18] S. J. Westrup and F.L.Harrison. *Collins Encyclopaedia of Music*. 1989.
- [19] S. Wolfram. *A New Kind of Science*. 2002.
- [20] M. Yee-King. Algorithmic Composition and the State of the Art. 2001.