STATE-BASED SEQUENCING: DIRECTING THE EVOLUTION OF MUSIC

Michael O. Jewell, Mark S. Nixon and Adam Prügel-Bennett
School of Electronics and Computer Science
University of Southampton, Highfield, Southampton, SO17 1BJ, UK

ABSTRACT

Traditional approaches to automatic music composition do not take into account the guided nature of music, instead augmenting existing material or generating scores based on provided seed parameters. Furthermore, these approaches often use a single algorithm to create a piece, where it is more natural to split the composition process into separate musical elements. Our new State-Based Sequencer counters both of these limitations, using an agent-based collection of algorithms combined with a technique to provide motivation to the resultant music. In this paper, we describe the methodology behind this system, and provide early results from an application of the framework.

1. INTRODUCTION

Algorithmic techniques have been used to create independent pieces of music using rule-based methods even without the need for computers. They were used as far back as 1026 when Guido d'Arezzo assigned pitches to vowel sounds[5] and, more recently, when Ron Pellegrino created music using light hitting wall-mounted photoresistors. However, as computers are now sufficiently powerful, algorithms are often carried out in software.

The burst of computational composing algorithms began with Arnold Schönberg at the start of the 20th century, with Webern and his successors forming serialism from these roots. Iannis Xenakis was a pioneer who, from his 'succès du scandale' *Metastaseis* in 1955, produced multimedia creations based on probability, sonic phenomena, texture, and random generation, and this work contributed to the stochastic approach of composition[3]. Further approaches, such as Voss and Clarke's fractal techniques[6], McAlpine's cellular automata method[5], and Burton's genetic algorithm systems[1] followed, and these make up a collection of 'stock' composition methodologies.

However, these existing techniques have typically only been applied to individual elements of musical composition. For example, given a melody and rhythm, an algorithm might be used to create suitable chord patterns. Without this *a priori* knowledge, it is difficult to provide a structure or direction to the resultant music. All human-composed music has an inherent sense of motivation, whether it be how a composer feels about a subject, or the purpose for which a piece is written, and this can be explored using new approaches.

Our new distributed approach treats the existing algorithmic techniques as building blocks for the creation of a music composition system, where different algorithms can be plugged in for evaluation, while constraining the generation through the use of a directing media. Rather than generating music with no prior information, a composer model[4] is used to provide *a priori* information for the algorithms, and a rich media ontology allows for the alteration of these parameters at pertinent points in the bound medium.

This paper is split into three key parts, which follow the process of composing under the State-Based Sequencer. The first describes the two representations that are required before composition can occur, namely the composer representation and the 'OntoMedia' ontological representation. The OntoMedia representation details the composing process itself and the agent framework that is used to provide the middleware for the distribution of composition tasks. We then discuss the individual agents, showing how each is tailored towards a certain aspect of the music composition, and finally give some preliminary results from some of the implemented agents and discuss how the system will be developed further.

2. PREPARING FOR COMPOSITION

The State-Based Sequencer requires two elements before it can proceed with composition: a marked up version of the directing media, and a set of composer mappings (see Figure 1). For the former, a representation conforming to the OntoMedia ontology is used, while the latter specifies the modifications that should be applied to the resultant music when certain conditions arise.

2.1. The OntoMedia Representation

OntoMedia is being developed parallel to SBS, and provides the capability to mark up any media, including film, radio, and books. The ontology supports the idea of timelines within a medium, as well as events that occur during these timelines. Entities participate in these events, and thus interactions between characters and objects can be labeled (see Figure 2 for a graphical representation of this). All events and entities are arranged hierarchically, with a set of core events (gain, lose, introduce, and transform) and a set of core entities (abstract item, physical item, character, and location) describing the base level of

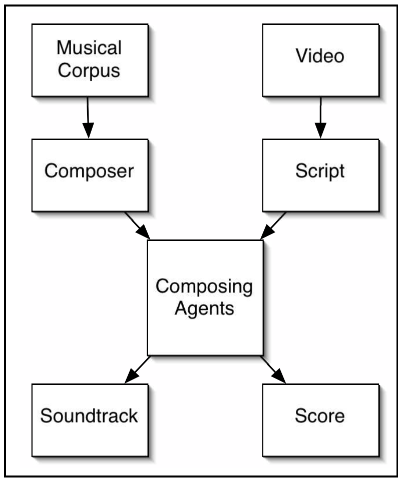

Figure 1. A high-level view of the State-Based Sequencer, showing the core elements of the framework.

information. For example, a character being threatened would be represented by a gain of a physical or a mental obstacle. This provides a significant amount of abstraction - it may be effective to move into a minor key whenever an obstacle is encountered.

Additionally, entities can have sets of actions that make up their 'motivations'. These specify, as is expected, the actions that they wish to occur. These could range from the gaining of an item (a subject finds a key) to a more complex combination (a subject escapes from a room and delivers an item). By examining the marked up medium, it is then possible to determine if, and when, a character's motivations are satisfied. This is a fundamental event, and often triggers a musical change.

Finally, the representation provides for detailed information regarding locations. By way of an example, the OntoMedia associated with a scene from the film The Matrix concerns Room 303 of the Heart's Hotel. The representation makes it possible to specify that the events occur in this room. Often the location is significant to the musical style that is selected, and this hierarchy allows a rich description of this information.

2.2. The Composer Representation

While the OntoMedia representation provides a formalised version of a directing medium, the composer representation establishes the link between the medium and the resultant music. A composer is defined as a set of pairs: the first element specifying a situation within the directing medium, and the second specifying what modifications should be made to the soundtrack.

As an example, it may be the case that a certain composer wishes to use saxophones whenever an event occurs within a room with blue walls. The situation would be defined in SeRQL (Sesame RDF Query Language), and would produce all events in which a room with blue walls occurs. The modification would specify that the probability of a saxophone being chosen as instrumentation should

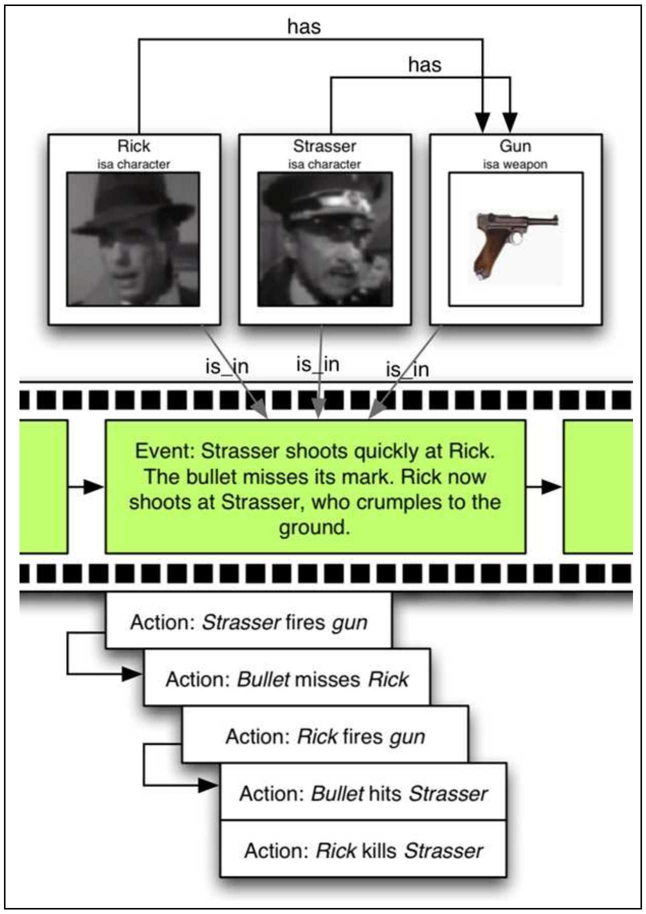
has
has
Rick
isa character
Strasser
isa character
Gun
isa weapon
is_in
is_in
is_in
Event: Strasser shoots quickly at Rick. The bullet misses its mark. Rick now shoots at Strasser, who crumples to the ground.
Action: Strasser fires gun
Action: Bullet misses Rick
Action: Rick fires gun
Action: Bullet hits Strasser
Action: Rick kills Strasser

Figure 2. An illustrated markup of a scene from Casablanca. Note the timeline containing important events, which are then subdivided into actions with participating entities.

be increased. The other parameters available for modification are described in the agent discussion in section 3.

The hierarchy inherent in the OntoMedia representation can be used to powerful effect in the composer representation. As stated earlier, a composer could introduce a minor key when obstacles are encountered, or an entire universe could have its own modifications. Having individual composer representations provides the ability to retain the character of human composers, and opens up possible future research into automatic representation creation from prior works.

The combination of the two representations allows for a translation of any media into a set of constraints that can be manipulated by a composition system, while ensuring that continuity is felt throughout the final piece.

3. THE AGENT FRAMEWORK

The composition system in SBS is built on top of the Light Agent Framework (LAF) middleware, with individual agents handling seperate composing tasks. The framework was created specifically to allow for simple agent implementations that can operate on a heterogeneous set of machines. At the centre of the framework is a router and, when an agent is created, it connects to this and is thereafter accessible by other connected agents or applications. Several agents can be available for a specific task and the router will lock the next unlocked agent of the type requested by the client. The selection process can also be customised, so it is possible to pick the next available agent on the node with the most resources.

In our system, every agent has a similar structure, shown in Figure 3. This consists of two input ports, two output ports, a parameter decoder, a result encoder, and the composing algorithm itself. All of the ports contain XML, with one input and one output corresponding to the landmark representation mentioned previously, and one input and one output corresponding to a MusicXML[2] representation of the music at that stage. MusicXML was chosen for two reasons, namely its ability to describe the content of music in high detail (including accents, clefs, and dynamic markings) and the availability of applications to convert from MusicXML to other formats, both audio and visual. MIDI, MuseData, and Humdrum were also considered, but are either less suitable for notation (in the case of MIDI) or harder to parse (in the case of MuseData and Humdrum).

The SBS composition system currently consists of seven agents: tempo, pulse, instrumentation, key, chord, rhythm, and melody. Of these seven, five use genetic algorithms to produce the final results. The two which do not are tempo and instrumentation, with tempo only using a genetic approach when no tempo is provided or when one cannot be easily calculated from event information, and instrumentation using a mean-based technique. The genetic approach to composing is recent, but is ideal for this situation as there is never a single ‘perfect’ piece of music. As such, having a population of scores represented as

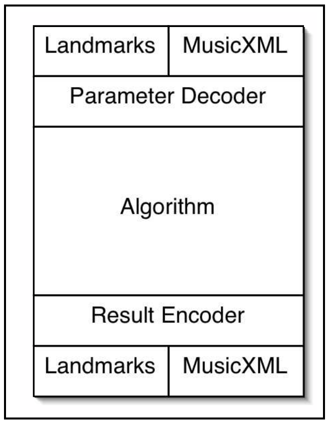

Figure 3. The standard structure used to represent a musical agent

chromosomes allows for greater flexibility, with a fitness value providing the results that are ‘most suitable’. The seven composing agents, as well as the remaining agents that are under development, are connected to one another in the arrangement shown in Figure 4 using an agent graph system. This monitors the input ports for required parameters and triggers the composing tasks automatically.

Providing the landmark and MusicXML ports at both the input and output stages allows for incremental adaptation of both the music and the parameters that are required. As is evident in Figure 4, it is possible to add detail to the music as the process progresses. Initially only tempo markings are available (via the Tempo Agent), but this is then built up by barlines (Pulse), staves for each part (Instrumentation), key signatures (Key), chord markings (Chord), and finally rhythmic information (Rhythm) and pitch information (Melody). Some of these are also easily parallelised, especially the earlier agents such as pulse, and the agent framework allows for several agents, each working on one part of the final score.

It is also necessary to alter the landmark information as the composition progresses, although only in a few cases. At the tempo generation stage all information (such as event locations) is in milliseconds, which is not an efficient measure for the successive agents. As such, the tempo agent converts the landmark information from milliseconds to beats, and passes this to the pulse agent. The landmark port also allows for the future addition of motific information in a motif agent.

3.1. Histogram-Based Agents

Four agents in the SBS framework make use of a histogram approach to fitness evaluation, namely tempo, pulse, instrumentation, and rhythm. In these cases the landmark representation contains a set of possible choices, each with a probability of occurring. To calculate the fitness of the chromosome, a value is determined which ensures that the options with the higher probabilities occur most often.

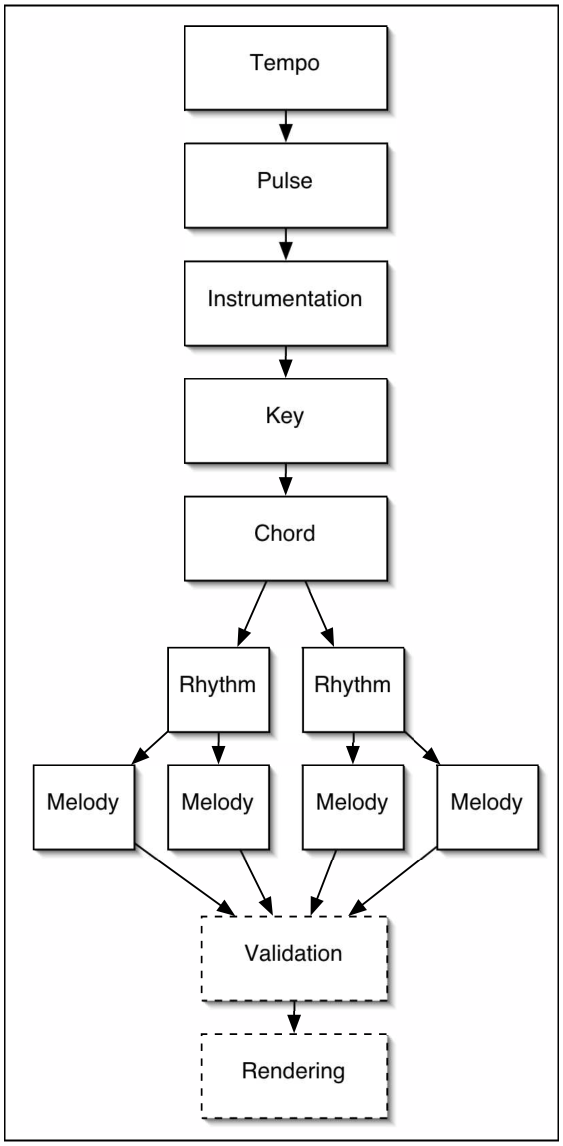

Figure 4. A structure for the distributed composition of music. Future agents have dashed borders.

3.2. Tempo Agent

3.2.1. Outline

The first agent in the composition process is responsible for specifying where tempo changes should be, and what beats per minute values should be used. This stage has less initial data than the others, as tempo is considerably more likely to rely on the source medium than on prior compositions for the material.

3.2.2. Implementation

The tempo agent makes use of the segmented structure of the landmark file. Each segment can be handled individually, and hence easier cases can be optimized for speed. First, segments that have exact tempos with no extra beats are processed. As it is not necessary to calculate the BPM, they can be immediately converted into separate landmarks (one per beat). Next, the agent handles segments that have no single tempo specified, but have beats in place. In this case, a set of non-unary factors is calculated where the members coincide with the beat locations on as many occasions as possible. For example, if beats occurred at frame 30, 40, and 55, this set would contain 5 as the best factor, followed by 10. If any of these values are present in the histogram, these are preferred, This is then converted into a beats per minute value.

The third case is where a fuzzy tempo has been suggested, i.e. ‘faster’ or ‘slower’. If any beats are present, these are first converted into a list of factors, as stated previously. If there is a factor that provides a BPM corresponding to the requirements, this is chosen. If no beats are present, a genetic approach is used. A chromosome is created with enough genes for the number of frames in the segment, with each gene containing a Boolean value specifying whether a beat is present in that location. A population is then created of random chromosomes, and this is evolved until the tempo stabilizes into a value present in the histogram.

Finally, segments with no tempo indications are handled. This case is handled with a straightforward approach:

1. If there is a preceding tempo indication, this is carried forward.

2. If there is no preceding tempo, but there is a succeeding tempo, this is brought backward.

3. If there is neither a preceding nor succeeding tempo, the most probable BPM in the histogram is used.

3.3. Pulse Agent

3.3.1. Outline

Once a tempo is available, the pulse agent makes use of the beat output combined with the pulse mappings for each segment to generate suitable strengths for each beat. The first beat of the bar (the downbeat) is where phrases usually begin, so it provides anchors for the succeeding agents. As the tempo agent reduces the landmarks from frames to beats, this agent is much more suited to a genetic algorithm, and so a form of pattern matching is used to provide fitness measures.

3.3.2. Implementation

The chromosome of the pulse agent genetic algorithm consists of a string of numbers, with each representing the strength of the beat at that position. Hence, $[1, 0, 1, 0]$ suggests the first two bars of a standard 2/4 beat. There are enough genes in each chromosome to accommodate the whole of the section, which can be quite large for higher tempos. As an estimate of the maximum, a 5 minute segment at 240bpm would have 1200 beats. However, the tempo is usually closer to 120bpm and shorter note lengths are used to suggest the faster speed.

Three operators are used in this agent, namely crossover, mutation and sequence. Crossover operates in the traditional style, with a range of genes being swapped between two chromosomes. Mutation is also based on the standard mutation operator, with a random gene being changed to a random value. The sequence operator, however, is unique to this system, and is used to provide repetition. A range is selected within the chromosome, from which a random non-unary factor is chosen. A string of genes of this length is then repeated throughout the section, with the genes selected from the start of the range. Repetition is fundamental to the pulse, so an operator to promote it is ideal in this situation.

The fitness function of the pulse agent is customized to guide the GA towards a suitable solution. Working from left to right, the beat options suggested by the composer representation are compared to a substring that is as long as the segment. For every gene that matches, $1/l$ is added to that option's score (where l is the length of the substring - so 1 is a perfect match). If one or more matches are exactly correct, the match that is most highly weighted is chosen, and the weight added to the score. Otherwise, the closest match is returned, and the score multiplied by the weight is added. Hence, if only half the string is matched, the score is halved accordingly.

3.3.3. Example

In this simple example, two options were available to the genetic algorithm: [1, 0, 0.25, 0.75] and [1, 0, 1, 0], both with equal weightings of 1. Applied to a chromosome of length 16, the pulse in Figure 5 was produced with fitness 60.

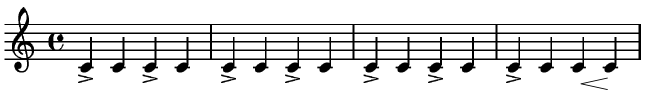

Figure 5. A basic pulse produced by the pulse agent. The accented beats indicate where the value of the pulse gene is 1, while the crescendo contains two beats with pulse values of 0.25 and 0.75.

Instrument	Weighting
Piano	6
Violin	5
Flute	5
Oboe	4
Guitar	2
Trumpet	1

Table 1. A simple instrument/weighting mapping for a segment.

3.4. Instrumentation Agent

3.4.1. Outline

The simplest agent in the framework, the instrumentation agent selects the number of parts in the score and which instruments should be assigned to each part. Each instrument in the system can be weighted, and this mapping is passed as a parameter to the agent.

3.4.2. Implementation

The implementation of the instrumentation agent is very simple. The landmark representation provides a set of instrument weightings, and these are accumulated. From this value the mean is found, and the instruments with weightings greater than this are chosen for the score. For example, given the instrument mapping in Table 1, $\overline{x} = 3.833$, piano, violin, flute, and oboe would be selected for the segment.

3.5. Rhythm Agent

3.5.1. Outline

The rhythm agent is the next logical step in the composition process. It uses the output from the pulse agent, and determines the note lengths to use in the segment. This agent is similar to the pulse agent in operation, again using a genetic algorithm to generate the final rhythm, but is applied to each instrument rather than to the score in general. This is the first case where the process can be parallelized by part rather than by segment, and is hence where the agent framework can be used to its full potential.

3.5.2. Implementation

The chromosome of the rhythm agent is closely tied to traditional note values. Each note length is assigned a double value between 0 and 1 (1 for a semibreve, 0.5 for a minim, and so on). The rhythm gene contains both this value and a Boolean to denote whether the note should be sounded or treated as a rest. The length of the chromosome is difficult to determine, as a rhythm that is entirely semiquavers would be 8 times longer than a rhythm comprised of minims. As such, a chromosome is reserved that can hold enough semiquavers to cover the segment. Unfortunately this can become very large, so a maximum size can be specified to reduce the overhead of the algorithm. To improve efficiency, only the section of the chromosome that is of a suitable duration is scored.

The rhythm agent uses a similar set of operators to the pulse agent, with the mutation altered to handle the 'rest' parameter of the rhythm gene. The fitness function is also based on the same approach of weighted pattern matching, although there are a few pertinent differences:

1. The duration of the chromosome is checked during the evaluation stage, and only the portion that is within the timescale of the pulse information is examined. This prevents the need to examine all of the genes within the chromosome.

2. The pattern weightings are multiplied by the value of the pulse at the start of the pattern. For example, if a pulse has a strength of 0.5 and the pattern in that location is weighted as 1.5, 0.75 is added to the final score. This guides the algorithm towards preferring rhythms that begin on strong beats of the bar, while allowing for some variation.

3.5.3. Example

Using the output from the pulse agent (see Figure 5, a generated pulse), a rhythm was generated. For this example, three options were available, namely [0.25, 0.125, 0.0625, 0.0625] (crotchet, quaver, semiquaver, semiquaver), [0.25, 0.125, 0.125] (crotchet, quaver, quaver), and [0.25, 0.0625, 0.0625, 0.125] (crotchet, semiquaver, semiquaver, quaver). The result of this test can be seen in figure 6.

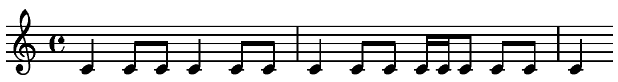

Figure 6. A rhythm generated using a set of rhythm seeds and the pulse from the pulse agent.

3.6. Markov-Based Agents

The final 3 'core' agents in the SBS system use a Markov approach to fitness evaluation. These are agents that are ideally suited to this technique, namely key, chord, and melody. Key and chord both have traditional rules for progression, such as relative major/minor for keys, and cadences for chords. Melody is less well-defined, but it is possible to initialise a graph of pitches based on existing music, or using scale information.

3.7. Key Agent

3.7.1. Outline

The key agent is the first in the framework to use a graph-based approach. While rhythm and pulse are simpler to specify as short cases, key changes are easier to represent as the probability of moving from one key to another. A scoring approach was created using this graph technique, which is at the core of the key agent fitness function.

3.7.2. Implementation

The first step in the key agent is to build up the key graph for the segment that is being evolved. A directed, weighted, graph is used, with each node being a key and each edge representing a key change. The key change landmark is read from the provided landmark parameter, and a graph is constructed with the correct weights on each edge. The weights used in this approach are simply double values, with no maximum or minimum. Even negative values can be specified, but it is preferable to use 0.

The chromosome for the key agent is simple, with each gene representing one of the keys available. For example, 'Cmaj' could be presented by 0, 'Dmaj' as 1, and so on. It is not necessary to encode more information into the gene, as the root and scale can be obtained from the landmark file when required. The chromosome length is set by the number of key changes required in the block, so if 10 key change landmarks are indicated, 10 genes are present in the chromosome. The traditional two operators (crossover and mutation) are used for this agent, with mutation setting the gene to a random key.

To make use of the graph data structure, a custom genetic algorithm was written. This moves by step through the chromosome and progress through the graph from node to node. If a weight is present on an edge, this score is added to the final total. One future option could be to allow for a nodal weighting to influence where the graph should begin, for example suggesting that C minor is preferable to C major.

3.8. Chord Agent

3.8.1. Outline

As with the key agent, a graph-based technique is used to develop the underlying chords for the score. These are essential to the melody evolution later in the process, as notes in the chord have priority over non-chordal notes. As with keys, there are some predefined rules for chord progression, which can be handled by the graph approach.

3.8.2. Implementation

Being similar in operation to the key agent, the chord agent first builds up a chord graph to represent the progressions available in the system. In this instance, the nodes in the graph refer to the unique names defined in the composer representation, such as 'I' or 'V7', and these are encoded as integers in the chromosome. The nodes are linked by the defined chord progressions, and an extra flag is present in the edge definition to specify whether the link can be a 'cadence' (an a edge linking the final two chords of a segment). The same operators are used as in the key agent, but in this case the mutation operator selects a random chord.

The fitness function uses the same technique as the key agent, but with one difference. If the edge between two chords is defined as a cadence, it only receives a score if it is at the end of the chromosome (or if the edge is defined twice - once as a cadence, once as a basic progression). This aims to finish the piece on a suitable ending.

3.9. Melody Agent

3.9.1. Outline

To create a score as efficiently as possible, the melody agent is also parallelised, with each agent creating the melodic line for a single instrument. As with the key and chord agents, a graph-based approach is used, although the melody agent will eventually also take motif information into account.

3.9.2. Implementation

As the pulse provided the 'skeleton' of the rhythm agent, the rhythm provides the framework to the melody agent. The available notes are further restricted by the pitch mesh of the segment and the instrument range, ensuring that the result is both suitable and playable. A chromosome is constructed with the same number of genes as there are notes in the rhythm output, with each gene containing an integer corresponding to the note value. These are held in MIDI format, which ranges from 0 to 127 (middle C being represented as 60). Each adjacent integer is one semitone higher than the previous, so C# is encoded as 61 and B as 59.

To score each chromosome, the fitness function begins with the initial note and locates its position in the pitch mesh. As it moves to the next note, it adds the weighting on the transition to its score. If no link is present between two notes, 0 is added. If a motif is required at a location, the pitch mesh is temporarily swapped out with the motif mesh, until the end of the motif is reached.

The melody agent makes use of the three operators defined by the pulse agent (crossover, mutation, and sequence), but also makes use of two others: step and inversion. The step operator is similar to the sequence, in that it repeats a segment of the chromosome over a length, but it also increases or decreases the pitch of the segment at each instance. The inversion operator is a simple operator that literally inverts a segment by subtracting each note value from 127.

3.9.3. Example

For this example, a mesh was initialised using Mozart's Ave Verum. This defined the probability of moving between pitch states, and hence the constraints for the fitness function. The agent was also passed the rhythm from Figure 6, and the results of this can be seen in Figure 7.

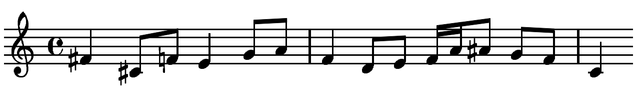

Figure 7. The output from the rhythm agent combined with a melody generated by the melody agent. Note that scale information has not been applied to the pitch mesh, hence producing some chromaticism.

3.10. Future Agents

There is scope for three extra agents to be added to the system once the core seven have been fully implemented, namely a validation agent, a rendering agent, and a motif agent. The first two of these (as seen in Figure 1) follow on directly from the melody agent, whereas the motif agent will work in parallel with the initial stages of composition.

3.10.1. Validation Agent

Once the initial composing process is complete, a large number of possible scores will be produced. The validation agent will be responsible for reducing the number of scores, as well as fine-tuning the final selection. The melody agent does not do any checks between parts, so the validation agent will check for missing notes, bad harmony (such as parallel fifths), and other such inconsistencies.

Where the other agents use a fairly lightweight genetic algorithm, the validation agent requires a far more intensive approach. The population will not be initially random, as is typically the case, with its members consisting of the results from the melody agents. The fitness function will then analyse on a score-by-score basis, providing a value based on the number of inconsistencies found within the piece. Operators will also be available to the algorithm, but again these will work on a score-wise basis. For example, the crossover operator must either exchange vertical slices (i.e. the same portion of every stave) or horizontal slices (two portions within the same stave).

The validation agent is a prime candidate for parallelisation, with seperate agents handling different combinations of scores, so it will be an ideal candidate for distribution under the agent framework.

3.10.2. Rendering Agent

The rendering agent is the final step in the composition process, and is very simple compared to the other components. This agent is responsible for collecting the final score, and producing a usable result. There is scope for several rendering agents, such as one for audio rendering and one for visual. Our use of MusicXML is beneficial in this stage, as it can be easily transformed to MIDI or PostScript/PDF.

3.10.3. *Motif Agent*

While the composition system may be able to produce music that is suitable for most media, it does not currently handle the idea of motific information. The OntoMedia representation provides the capability to identify major ideas, so a motif agent will use this to generate themes. These will be very generic, as they must be easily placed into different sections of the music, where the key, chords, rhythm, or tempo may all be different. This also allows for motific change based on character information—so a 'bad guy' may have a minor motif.

4. CONCLUSIONS

The SBS system of composing investigates a novel technique in coupling media with music composition, and at this stage in the project results are very encouraging. The provision of a media description ensures that the composed music is suitable for the events that occur, with the composer representation allowing for different interpretations of events and concepts. Furthermore, by separating the process into several agents the approach is highly analogous to the traditional composing style, as well as being easily distributed over a set of machines to reduce processing time. By moving away from monolithic approaches to score generation, and instead taking advantage of the inherent hierarchy of music, our system is both simple to test and easily updated.

Two further agents, one for music validation and one for music rendering, are already at the design stage, so we hope to obtain complete generated scores for further testing. The validation stage is particularly suitable for the agent framework, as we plan to generate a large number of possible scores and use these as the bootstrap for a genetic algorithm. This will require tens, if not hundreds, of melodies, and hence will involve many melody agents communicating with several accumulation agents, with these communicating with the validation agent.

5. REFERENCES

[1] Burton, A. R., and Vladimirova, T. Generation of musical sequences with genetic techniques. *Computer Music Journal 23*, 4 (1999), 59–73.

[2] Good, M. MusicXML: An internet-friendly format for sheet music. In *XML Conference and Expo* (2001).

[3] Harley, J. The electroacoustic music of iannis xenakis. *Computer Music Journal 26*, 1 (2004), 33–57.

[4] Jewell, M. O., Nixon, M. S., and Prügel-Bennett, A. CBS: A concept-based sequencer for soundtrack composition. In *WEDELMUSIC* (2003).

[5] McAlpine, K., Miranda, E., and Hoggar, S. Making music with algorithms: A case-study system. *Computer Music Journal 23*, 2 (1999), 19–30.

[6] Voss, R. F., and Clarke, J. 1/f noise in music and speech. *Nature 258* (1975), 23–33.