Anticipatory Timing in Algorithmic Rhythm Generation

Toby Gifford *Andrew R. Brown*

Queensland University of Technology
Victoria Park Rd, Kelvin Grove, Australia

1. Abstract

Generative music algorithms frequently operate by making musical decisions in a sequence, with each step of the sequence incorporating the local musical context in the decision process. The context is generally a short window of *past* musical actions. What is *not* generally included in the context is *future* actions. For real-time systems this is because the future is unknown. Offline systems also frequently utilise causal algorithms either for reasons of efficiency [1] or to simulate perceptual constraints [2]. However, even real-time agents *can* incorporate knowledge of their own future actions by utilising some form of planning. We argue that for rhythmic generation the incorporation of a limited form of planning - *anticipatory timing* - offers a worthwhile trade-off between musical salience and efficiency.

We give an example of a real-time generative agent - the Jambot - that utilises anticipatory timing for rhythmic generation. We describe its operation, and compare its output with and without anticipatory timing.

2. Introduction

This research has been conducted in the context of a larger research program investigating real-time machine listening and improvisation, culminating in a real-time machine improvisation agent, the Jambot. In this paper we discuss the algorithm used by the Jambot for generating improvised percussive rhythms.
This work extends previous work by the authors in the Ambidrum [3], a system for generating percussive rhythmic material with a specifiable degree of metric ambiguity. The primary differences are the inclusion of many more rhythmic heuristics, and an extended search technique that we are calling *anticipatory timing*.

Anticipatory timing is an extension of the typical approaches to searching for generative algorithms that operate by optimising a musical fitness function at each step of a sequence of decisions. It involves extending the search to look a short way into the future, by searching over both the best note to play next, and the best time to play it.

In sections 3 - 5 we discuss existing approaches using contextual models for algorithmic music generation and analysis. The approaches are divided into two categories: rule-based systems and statistical models. The two categories are related, in that a rule-based system can be transformed into a statistical model and visa-versa. It is convenient for our purposes, however, to think in terms of rule-based systems because of their natural interpretation in terms of mathematical optimisation.

Considering algorithmic music generation as a problem in optimisation brings to the fore the question of efficient search techniques. Efficiency is a critical concern, even for non-real-time algorithms, because the size of the search space is vast due to the exponential growth of musical possibilities with the number of notes considered.

Rule-based systems have utilised a number of techniques to search the space of possible actions more efficiently than a naive brute force search. Many of these techniques involve performing the search in a step-by-step fashion, with the addition of some heuristics to 'prune' unlikely branches from the search tree.

The most extreme form of pruning is the class of so-called 'greedy' search algorithms [4], which hope to create an optimal sequence by making a series of locally optimal choices. Often the resulting sequence is far from globally optimal, however the search is extremely fast.

Real-time systems are subject to much greater efficiency constraints than offline systems. Moreover, the nature of real-time systems demands some form of sequential processing. Consequently greedy search algorithms are very common in real-time improvisation systems.

The idea behind *anticipatory timing* is that it provides a mechanism to enhance greedy search to find better optima, with a tolerable increase in computational complexity. We suggest that anticipatory timing is particularly efficacious for the modelling of rhythm, which involves a mixture of pattern based modelling with a need to align the timing of elements of the pattern with both previous elements of the pattern, and with contextual cues such as the underlying metre.

In section 10 we describe a new generative model of rhythm that is incorporated into the Jambot. This model is a rule-based system that combines a number of rhythmic heuristics, including heuristics regarding the relation of the rhythm with the underlying metre, and heuristics that describe surface properties of the onset timings irrespective of the metre.

Gifford, T. and A. R. Brown. 2010. Anticipatory Timing In Algorithmic Rhythm Generation. In *Australasian Computer Music Conference, Canberra.* ed. T. Opie, 21-28: Australasian Computer Music Association.

We give some examples of the Jambot's rhythm generation algorithm, both in solo and ensemble contexts. We compare the output with and without the use of anticipatory timing in the search, and conclude that the use of anticipatory timing enhances the musical coherence of its output.

The audio examples may be downloaded from http://dr.offig.com/research/papers/ACMC2010/AudioExamples.zip

3. Previous Work

This work extends previous work by the authors in the Ambidrum [3]. The Ambidrum is a system for generating percussive rhythmic material with a specifiable degree of metric ambiguity. It assumes that an underlying metre is supplied, and operates in real-time by applying heuristic rules relating the metric strength of the current beat to the emphasis of the note in the dimensions of velocity, duration and timbre. The generation of notes is performed at each beat (of a quantised time-line) by conducting an optimisation over the competing heuristics.

The Jambot's rhythm generation extends the Ambidrum by adding a number of other heuristics, including rhythmic properties that are not related to the metre, but reflect patterns of recurrence in the onset times.

The Ambidrum produces notes of a variety of durations, often longer than a single quantised beat. However, the underlying sequence that it uses is based on computing at regular time slices rather than generating one note after the duration of the previous has passed as is common in offline melody generation algorithms. In the Ambidrum process durations are implemented using a suppression technique; at each time slice the rules first check if the previous note's duration has finished, and if not the system takes no action.

The Ambidrum chooses its musical actions at each time-step by searching over the space of possible notes for the best note at that time. This type of process is known as a greedy algorithm (discussed further in section 7) because it opts for the best possible choice now, without any consideration of the future.

It is capable of producing simple rhythms that align closely to the metre, such as a standard rock beat. It is also capable of producing more complex rhythms that do not align with the metre, however it is difficult to control these more complex rhythms.

The added heuristics in the Jambot's rhythm generation allow for much greater control of complex rhythms by mixing rules that target metric ambiguity with rules that target other forms of rhythmic coherence. By adding these extra constraints, complex rhythms can be produced by deliberate departures from the metre that nevertheless retain musical coherence.

3.1 Context Models

Most generative music algorithms, particularly real-time algorithms, operate in a sequential fashion. The steps in the sequence are most commonly either regular time intervals (for example as in a step sequencing drum machine) or notes (as is typical of common practice notation). At each step the algorithm must make a decision as to what (if any) musical action to take. Generally algorithms will incorporate the local musical context into their decision making process.

The context is usually taken to be the immediate history of musical material:

> The idea behind context models, which we are mostly interested in here, is that events in a musical piece can be predicted from the sequence of preceding events. [5]

The notion that musical events can be predicted from a history of previous events has led to a multitude of generative algorithms that can be categorised into two broad categories:

i. Rule-based systems - optimisation based approaches that combine heuristic rules with observed history to create a fitness function for the space of musical actions. The note to be played at each step is the note with the highest fitness.
ii. Statistical models - stochastic systems that provide a probability distribution for the space of musical actions. At each step the note to be played is drawn randomly from this distribution.

These two categories are related; a rule-based system can be transformed into a statistical model by reinterpreting the fitness function as a probability distribution, and a statistical model may be transformed into a rule-based optimisation system by choosing the maximum-likelyhood value at each step rather than making a random selection [6].

3.2 Rule-Based Systems

Rule-based systems for describing music operate by supplying a collection of heuristic properties that music is expected to obey. Historically, many rule-based systems have been intended for analysis rather than for generation [7, 8]. However, if systems for analysis are formalised sufficiently to afford computational implementation, then they may be used 'in reverse' as generative algorithms [9, 10].

A frequently cited example of a rule-based system for musical analysis is the Generative Theory of Tonal Music (GTTM) [11]. The GTTM supplies a collection of preference rules that well-formed music should conform to. Actually performing an analysis using the GTTM involves coming to a compromise between competing preference rules. Lerdahl & Jackendoff's model stops short of providing a computationally implementable scheme for deciding between competing rules.

> The reason [that we have not implemented our theory computationally] is that we have not completely characterized what happens when two preference rules come into conflict. Sometimes the outcome is a vague or ambiguous intuition ... We suggested above the possibility of quantifying rule strengths, so that the nature of a judgment in a conflicting situation could be determined numerically. [11:54]

There have been various efforts to provide computational implementations of the GTTM or similar preference rule systems [12, 2]. These implementations consider the problem as a question of *optimisation*. In other words the correct analysis is found by searching through the space of possible analyses for the best analysis according to some weighted combination of the rules.

> In a given preference rule system, all possible analyses of a piece are considered. Following Lerdahl & Jackendoff, the set of "possible" analyses is define by basic "well-formedness rules". Each preference rule then assigns a numerical score to each analysis ... The preferred analysis is the one with the highest score. [2:15]

The task of implementing a computational version of the GTTM (or any rule-based system) as an optimisation problem is made difficult by vast number of possible analyses to be searched over for any given piece of music. To perform a naive search over all analyses in search of the optimal one is generally computationally intractable. There are, however, a number of search techniques that are substantially more efficient than the 'brute force' approach of exhaustive search.

A common approach to dealing with the combinatorial explosion of searching over all possible analyses is to perform a search in a sequential fashion, and to use some heuristics for 'pruning' unlikely branches of the search tree. Such approaches also seem more likely to provide a model for "the moment to moment course of processing as it unfolds during listening" [2:14]. One such approach is dynamic programming [13] which has found use in a number of algorithmic analysis and generation systems [14, 15, 16, 17, 2, 1].

Temperley has constructed a preference rule system similar to the GTTM called the Melisma model that affords computational implementation, and utilises dynamic programming to search for the 'best' analysis for a given piece of music.

> When the program searches for a globally optimal analysis based on the results of the local analyses, it must consider global analyses rather than simply choosing the best analysis for each segment in isolation ... Usually, even for a monophonic short melody, the number of possible local analyses may grow exponentially with the number of segments, and the size of the best-so-far analysis becomes extremely large. The Melisma system suppresses the explosion of analyses by properly pruning less significant analyses by dynamic programming. Temperley argues that, to some extent, this searching process reflects the human moment-to-moment cognitive process of revision, ambiguity and expectation when listening to music [12]

The Melisma system, although originally intended for computational analysis, has been 'reversed' to be a generative algorithm, the Melisma Stochastic Melody Generator [18], providing both an example of an analytic theory transformed into a generative process, and an example of a (deterministic) rule based system reinterpreted as a stochastic statistical model.

3.3 Statistical Models

There have been many statistical models of music generation, from simple mapping of mathematical functions such as periodic motion and random walks, to the use of more complex processes such as chaos theory and cellular automata. Some of the more musically successful statistical models have been probabilistic, often based on probability distributions derived from analysis of previously composed material. Recent coverage of probabilistic tendencies in music have been written by Huron [20] and Temperley [21]. Whilst these publications have been focused on music analysis the use of these approaches for music generation is widely acknowledged, as described by Conklin:

> Analytic statistical models have an objective goal which is to assign high probability to new pieces in a style. These models can guide the generation process by evaluating candidate generations and ruling out those with low probabilities. The generation of music is thereby equated with the problem of sampling from a statistical model, or equivalently, exploring a search space with the statistical model used for evaluation. [10]

In a simplistic sense statistical models can be used to generate musical material at each step in sequence independently of context, however, as Conklin again acknowledges, it is also common and often more effective to use previous and concurrent note events to inform statistical processes of generation.

> The most prevalent type of statistical model encountered for music, both for analysis and synthesis, are models which assign probabilities to events conditioned only on earlier events in the sequence. [10]

The most common type of probabilistic models that uses past events as context are Markov models. These have been used for generation of computer music since Hiller and Isaacson's compositions in the late 1950s. Markov models are useful for sequential processes such as music because they describe the frequency of sequences of events, such as which rhythmic values follow each other.

Markov models, and many other statistical approaches, while they take account of past context, do not take account of future possibilities and therefore, we suggest miss an important opportunity to produce more optimal event selections. One challenge for extending statistical models to consider multiple future scenarios is the computational complexity of that can result.

3.4 Greedy Algorithms

Many real-time generative systems that utilise either Markov models or rule based operation operate as what is described as a greedy algorithm [4:370]. In an optimisation context a greedy algorithm is one which generates each step of the sequence by optimising locally. Generally greedy algorithms fail to find the global optima [21]. However, they are often used because they are much faster than exhaustive searches.

> The random walk method, while applicable for real- time music improvisation systems that require fast and immediate system response (Assayag et al., 1999; Pachet, 2002), is flawed for generating complete pieces because it is "greedy" and cannot guarantee that pieces with high overall probability will be produced. The method may generate high probability events but may at some stage find that subsequently only low probability events are possible, or equivalently, that the distribution at subsequent stages have high entropy. [10]

Greedy algorithms represent the most extreme form of 'pruning' of the search tree; only the current action is searched over – which amounts to pruning *all* of the branches of the tree, leaving only the current node.

3.5 Existing Approaches to Rhythm Generation

There have been a number of approaches to the creation of rhythmic material in generative music research. We will describe some of them here so as to assist with comparison with our anticipatory timing approach.

A number of projects simply utilise existing or fixed rhythmic material for their generative music and thus focus attention on the pitch and or harmonic generative processes. Such systems include the Continuator [22] which replays rhythms played into it, and computational creativity experiments by Pearce & Wiggins [23] that utilise human composed rhythmic material and vary the pitched material.

In a more granular way systems such as Cope's EMI [24] re-use sampled rhythmic material and rhythmic structure rather than generating it from rules.

Generative rhythmic systems include those that use onset probabilities at each time slice as described by [20:30].

Oscillating Rhythms [25] uses cyclic functions to achieve patterns of recurrence of various rhythmic dimensions (onset, velocity and timbre).

OMax [5] uses a predictive coding structure trained on a database of rhythmic features, including duration, to decide on the next most likely note.

OMax relies time based sequencing using a greedy algorithm, however it just uses probabilities of the current temporal location and does not employ any look ahead [5].

Collins [1] uses dynamic programming for his offline music generation algorithm. He also provides a greedy online version as an option for faster calculation.

4. Anticipatory Timing

4.1 Definition of Anticipatory Timing

To incorporate planning by naively searching over even a short number of future actions is computationally intractable. Even for offline systems this is not feasible, so smarter approaches to searching, such as dynamic programming, must be employed. For real-time systems such as we are concerned with computational efficiency is a strong constraint, and even full dynamic programming may be too expensive, so that greedy online variants of dynamic programming are sometimes used [1]

We propose *anticipatory timing* as a computationally tractable means of improving upon greedy algorithms.

Anticipatory timing is an extension of greedy optimisation to include some level of anticipation in the timing of the next action. This involves searching both over possible actions *and possible times* for the next action. At each step of the optimisation, the fitness function is calculated for each possible action, and recalculated again at each time slice (for a short window of the future). This calculation is done with the constraint that only one action is considered. If the highest overall value for the optimisation occurred at the current time slice, then that action is taken. Otherwise no action is taken at the current time slice.

If the algorithm anticipates that the best time for action is a few steps into the future then it does nothing at the current time slice. Supposing that nothing else changes (in the environment for example) then at that future time the action is likely to be taken. However, if in the meantime something unexpected has happened, then the algorithm may re-asses the situation and decide against taking action at this time. This is particularly important in a real-time improvisation context, because it allows for *planning without commitment,* so that the algorithm may be flexible to unexpected changes in the improvisation.

4.2 Interpretation as pruning the search tree

Anticipatory timing is similar to greedy optimisation in only planning a single action at a time, but allows for the possibility of adapting the timing of this event if required. At each step of the generation the optimisation routine examines future outcomes at each time slice until the next event, but not every possible outcome in the rhythmic space. This increases the computational demand linearly with the number of beats that we look ahead. This compares favourably with the exponential increase of complexity in planning a number of actions ahead.

In terms of 'pruning' the search tree, anticipatory timing amounts to pruning down to just a single branch. This branch consists of a series of 'take-no-action' nodes and terminates when an action is taken. This means that the number of computations required overall is equal to the number required for searching over a single action multiplied by the number of time slices into the future that we look.

4.3 Comparison with full search

An alternative to anticipatory timing would be to plan a number of notes in advance. Presuming that we have a fixed amount of computational power available, a quick computation demonstrates how much further into the future we can look using anticipatory timing, rather than searching over the entire tree.

Suppose, for purposes of demonstration, that there are 16 possible notes to consider at each time slice. Suppose further that the bar is divided into 16 semi-quaver time slices. Using anticipatory timing we look a full bar ahead: i.e. at each of the 16 time slices in the bar we search over 16 notes. Then the total computation involves searching over 16 x 16 = 256 notes.

On the other hand, consider attempting to plan some number of notes into the future considering all possible combinations. Each time slice we must search over 16 notes, and for each of these notes we must search over 16 notes at the next time slice, and so on. Then by the second time slice we must examine 16 x 16 = 256 notes.

So for the same computation budget, we can look ahead a full bar of 16 semiquavers with anticipatory timing, but can only look *one* semiquaver ahead if we attempt to search the entire tree. To look ahead the full bar for all possible notes would require searching over 16^16 notes, which is approximately the number of grains of sand in the earth [26].

4.4 The Chess Analogy

Lets consider an analogy with chess. In playing chess it is generally important to think as many moves ahead as possible. Computer chess programs such as Deep Blue have typically required vast amounts of computing power to be able to do this because of the enormous number of states required to search over. The problem arises because the number of states grows exponentially with the number of moves ahead to be searched. Even supercomputers like Deep Blue do not attempt to naively search over the entire tree of future moves, but use heuristics to prune the tree to a manageable size:

> At the heart of Deep Blue's ability to play chess is its evaluation function. The evaluation function is an algorithm that measures the "goodness" of a given chess position. Positions with positive values are good for White, and conversely, positions with negative values are good for Black. If the overall score is negative, for example, this means that Black has the advantage ... Deep Blue employs a system called selective extensions to examine chessboard positions. Selective extensions allow the computer to more efficiently search deeply into critical board arrangements. Instead of attempting to conduct an exhaustive "brute force" search into every possible position, Deep Blue selectively chooses distinct paths to follow, eliminating irrelevant searches in the process. [27]

This is a similar problem to the one faced by a rule-based generative algorithm in attempting to incorporate planning: to attempt to search over the entire tree of future actions is computationally infeasible for even a small number of future

actions. If planning is to be incorporated there must be some mechanism for pruning the tree. Deep Blue's approach is a form of online greedy optimisation with forward thinking; the evaluation function is akin to the heuristic rules.

The solution that we propose - anticipatory timing - involves draconian pruning indeed. As we discussed above, anticipatory timing only plans one action ahead, but allows for finessing of the timing of this action. In the analogy of the chess game, this corresponds to thinking just one move ahead - but considering the possibility of 'passing' until the best time to pull of this masterstroke. As it happens the rules of chess do not allow players to pass, so the analogy fails in this respect. However, as chess players can attest, at certain times it would be advantageous to pass. In terms of pruning the search tree, this corresponds to searching just a single branch of the tree, i.e., the branch that takes no action until an optimal time and then makes a single move.

5. Examples using the Jambot

5.1 The Jambot's Rhythm Generation

The heuristics used in the Jambot's rhythm generation are a combination of rules that relate the accent structure of the rhythm to the metre, and rules that target patterns of recurrence in the onset times of the rhythm.

The rules that relate the rhythmic structure to the metre include the rules used by the Ambidrum [3] for targeting metrical ambiguity, and additionally include a measure of syncopation - defined as occurring when a note's duration extends through a beat of greater metrical strength than that of the note's onset.

The rules that relate purely to the surface rhythmic structure, regardless of the metre, include a measure of the frequency of a given inter-onset-interval (defined as the difference in onset times between any pair of notes, not just successive notes) and a simple measure of the density of the rhythm.

5.1 Offbeat snare

The anticipatory timing approach that uses this look-ahead procedure seems to be most helpful when constructing rhythms that utilise a mixture of the metrical and non-metrical heuristics.

The anticipatory timing approach that uses this look-ahead procedure seems to be most helpful when constructing rhythms that utilise a mixture of the metrical and non-metrical heuristics.

As a first example, consider trying to construct a rhythm in 4/4 quantised to 8 quavers, consisting of a kick on the downbeat, hi-hats on each crotchet beat alternating with snares on each quaver offbeat.

The heuristics that we use for the hi-hat are density, syncopation and a periodicity of 2 quavers. The density is set to 50%, since we want high hats on the crotchet beats, but not the quaver offbeats. The syncopation is set to 0% since we want the hi-hats to be on all the on-beats. Finally, we set a large weighting towards a periodicity of 2 quavers.

For the snare we use similar heuristics, except we target a syncopation of 100%, so as to force the snare drum to sound on the offbeats.

The resulting rhythm, using look-ahead, is shown below in figure 1. and can be heard in the example file: *AlternatingSnareAndHatAnticipation.mp3*.

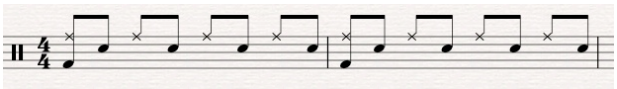

Figure 1: Offbeat snare hits with anticipatory timing

However, using the same settings for the heuristics but with the look-ahead turned off results in the music shown in figure 2. and heard in the example file: *AlternatingSnareAndHatNoAnticipation.mp3*.

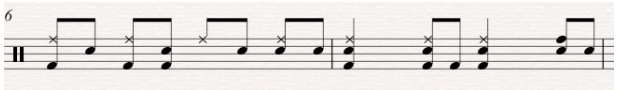

Figure 2: Offbeat snare hits without anticipatory timing

Without the look-ahead function the hi-hat pattern remains intact, but the snare pattern is all over the place. One way to think about what is happening here is that the offbeat alignment is being forced by syncopation, but without the look-ahead the algorithm can't tell that a given offbeat is a good opportunity for a syncopation because it's not looking ahead to the stronger on-beat.

5.2 Dancehall kick pattern

As another example we try to construct a 'dancehall' pattern with the kick drum, being a 3 + 3 + 2 rhythm in quavers in 4/4. The heuristics that we use are a combination of periodicities of 3 and 8. The periodicity of 3 on it's own would tend to produce a stream of 3 + 3 + 3 ... indefinitely. However, the addition of a periodicity of 8 constrains the rhythm to attempt to repeat itself every 8 quavers, and so we hope that this combination of settings should tend to produce the desired pattern, or a permutation of it (i.e., 3 + 2 + 3). To obtain the desired permutation, and have it align to the metre, we also specify the metric ambiguity heuristic to be low, meaning that the rhythm should adhere to the metre as closely as possible. The resulting rhythm (with a cow bell and clave click to show the metre) with the look-ahead turned on is shown in figure 3. and can be heard in the example file:
DanceHallKickAnticipation.aif.

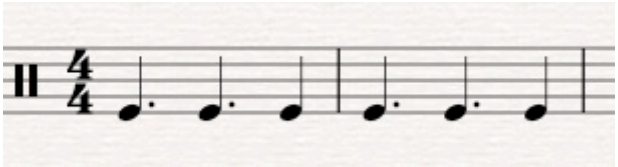

Figure 3. Dancehall beat with anticipatory timing

To see the effect of the look-ahead, note that the same settings with the look-ahead turned off gives the result shown in figure 4 and heard in the example file:
DanceHallKickNoAnticipation.aif.

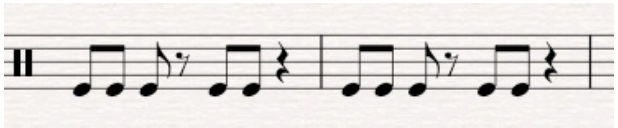

Figure 4. Dancehall beat without anticipatory timing

Without anticipatory timing the 3 + 3 + 2 pattern is lost.

5.3 Ensemble Interaction

In the examples above the context for the Jambot's rhythm model has been a short window of history of its own musical actions. The Jambot is, however, primarily designed to be used for improvising in an ensemble, in which case the context consists of both its own musical actions and those of the ensemble.

As an example of this mode of interaction we had the Jambot improvise in real-time to a pre-recorded drum loop, the infamous Amen Break, shown in figure 5, and heard in the example file *AmenBreak.mp3*.

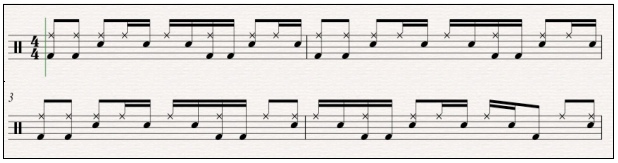

Figure 5. The Amen Break

The heuristic settings used were just the density and syncopation for each of the hi-hat, snare and drum. The settings used were:

Hi-hat: density 95% syncopation 0%
Snare: density 33% syncopation 50%
Kick: density 33% syncopation 0%

The resulting rhythmic improvisations with anticipatory timing is given in *EnsembleAnticipation.mp3*. The Jambot's improvisation is relatively sparse, consisting of a repeated pattern of a few snare hits only. This is because the Hi-hat and the Kick parts in the loop are already sufficiently busy to satisfy the density target.
The snare pattern, to our ears, complements the input rhythm in a musically appropriate and coherent fashion.

The same settings without anticipatory timing produced the result heard in *EnsembleNoAnticipation.mp3*. As with the previous improvisation it is restricted to the snare. The snare pattern in this example is, to our ears, still interesting but less musically coherent than the example with anticipatory timing.

6. Conclusion

We have described a method for assisting rule-based generative rhythm algorithms to produce more musical outcomes by anticipating possible future events and using this information as a constraint on committing to note generation (or early termination). We have examined previous computational approaches to generative rhythmic material and seen that the use of future context is very rare. This is somewhat understandable given the exponential increase in computational expense normally associated with searching future outcomes. This is a complex problem faced by many tasks, not just music, as we can see in the case of the Deep Blue chess playing system.

Our approach to anticipatory timing uses substantial search tree pruning and limits look-ahead to one note, such that it allows for this approach to be practical in real-time generative music systems. We have provided examples of an implementation of this approach in the Jambot interactive music system that provide data that allow us to make provisional claims about the success of this approach in practical musical contexts. We conclude that the use of anticipatory timing provides an effective means for increasing musical salience with a tolerable increase in computational complexity.

7. References

[1] Collins, N. "Infno: Generating Synth Pop and Electronic Dance Music on Demand", *in Proceedings of the International Computer Music Conference*. Belfast, UK, 2008

[2] Temperley, D. *The Cognition of Basic Musical Structures*. Cambridge, MA: The MIT Press, 2001.

[3] Gifford, T & Brown, A. R. "The Ambidrum: Ambiguous generative rhythms", *in Proceedings of the Australasian Computer Music Conference*, Adelaide, Australia, 2006

[4] Cormen, T., Leiserson, C., Rivest, R., Stein, C. *Introduction to Algorithms*. Cambridge MA: The MIT Press, 2006.

[5] Assayag, G. & Dubnov, S. "Using Factor Oracles for Machine Improvisation", *Soft Computing* 8:1-7, 2004

[6] Barto, A. G, Bradtke, S. J. & Singh, P. S. "Learning to act using real-time dynamic programming". *Artificial Intelligence* 72:81-138, 1995.

[7] Schenker, H. *Harmony*. Chicago: Chicago University Press, 1980.

[8] Narmour, E. *The Analysis and Cognition of Basic Musical Structures*. Chicago: Chicago University Press, 1990.

[9] Brown, A. R., Gifford, T., Davidson, R. & Narmour, E. "Generation in Context" *in the Proceedings of the 2nd International Conference on Music Communication Science,* Sydney, Australia, 2009

[10] Conklin, D. "Music Generation from Statistical Models" *in AISB Symposium on Artificial Intelligence and Creativity the Arts and Sciences*, 2003

[11] Lerdahl, F. & Jackendoff, R. *A Generative Theory of Tonal Music*. Cambridge, MA: The MIT Press, 1983

[12] Hamakana, M., Hirata, K. & Tojo, S. "Implementing a Generative Theory of Tonal Music" *Journal of New Music Research*, 35(4):249-277, 2006

[13] Bellman, R. & Kalaba, R. *Dynamic programming and modern control theory*. Academic Press, New York, 1956.

[14] Rowe, R. *Interactive Music Systems*. Cambridge, MA: The MIT Press, 1993.

[15] Pennycook, B., Stammen, D.R. & Reynolds, D. "Towards a computer model of a jazz improviser" *in Proceedings of the International Computer Music Conference,* San Francisco, USA, 1993 pp. 228-231

[16] Dannenberg, R. "Dynamic Programming for Interactive Music Systems". in Mirando, E. (ed) *Readings in Music and Artificial Intelligence*, 2000.

[17] Rolland, P.R. & Ganascia, J.G. "Musical Pattern Matching and Similarity Assessment". in Mirando, E. (ed) *Readings in Music and Artificial Intelligence*, 2000.

[18] Temperely, D. and Sleator, D. *Melisma Stochastic Melody Generator*. http://www.link.cs.cmu.edu/melody-generator/. Accessed 9 May 2010

Gifford, T. and A. R. Brown. 2010. Anticipatory Timing In Algorithmic Rhythm Generation. In *Australasian Computer Music Conference, Canberra.* ed. T. Opie, 21-28: Australasian Computer Music Association.

[19] Huron, D. *Sweet Anticipation*. Cambridge, MA: The MIT Press, 2006.

[20] Temperley, D. *Music and Probability*. Cambridge, MA: The MIT Press, 2007.

[21] Rusell, S. & Norvig, P. *Artificial Intelligence: A modern approach*. New Jersey, USA: Prentice Hall, 2003/

[22] Pachet, F. "Interacting with a Musical Learning System: The Continuator" *in Music and Artificial Intelligence*. Berlin: Springer, 2002 pp. 103 – 108.

[23] Pearce, M. & Wiggins, G. "Evaluating Cognitive Models of Music Composition" *in Proceedings of the 4th International Joint Workshop on Computational Creativity,* 2007

[24] Cope, D. *Experiments in Musical Intelligence*. A-R Editions, 1996.

[25] Sorensen, A. *Oscillating Rhythms*. http://www.acid.net.au/index.php?option=com_content&task=view&id=99. Accessed 9th May 2010.

[26] Argonne National Laboratory, *Ask A Scientist*. http://www.newton.dep.anl.gov/askasci/ast99/ast99215.htm. Accessed 9th May 2010.

[27] IBM Research. *How Deep Blue Works: Under the hood of IBM's chess playing supercomputer*. http://www.research.ibm.com/deepblue/meet/html/d.3.2.html. Accessed 9 May 2010.

Gifford, T. and A. R. Brown. 2010. Anticipatory Timing In Algorithmic Rhythm Generation. In *Australasian Computer Music Conference, Canberra.* ed. T. Opie, 21-28: Australasian Computer Music Association.