Beyond the Transition Matrix: A Language-Independent, String-Based Input Notation for Incomplete, Multiple-Order, Static Markov Transition Values

Christopher Ariza

ariza@flexatone.net

Abstract: While Markov chain generators have been employed throughout the history of computer music as a tool for the creation of musical parameter values, input notations for Markov transition values are often cumbersome and opaque. Rejecting the transition matrix as an input notation, this paper offers a new language-independent, string-based input notation for incomplete, multiple-order, static Markov transition values. Transition values are given greater generality by accommodating multiple orders simultaneously, as well as the specification of transitions with the use of limited single-operator regular expressions. A complete Python implementation of this model is introduced, and high-level utilities and object interfaces are demonstrated in athenaCL.

1. Introduction

Throughout the history of computer music Markov chain generators have been employed as a tool for the creation of musical parameter values. There are two common approaches to configuring these Markov processors. In the first case, a data sequence is analyzed, and then the resulting transition data is stored in a transition matrix and used to generate new values. In the second case, transition values are directly specified without derivation from analysis. In both cases, an intuitive and transparent notation of transition values, beyond the common tabular matrix, is advantageous. In the second case, a practical notation for Markov transition values offers a powerful and efficient input notation for a wide range of Markov applications.

Rejecting the transition matrix as an input notation, this paper offers a language-independent, string-based input notation for incomplete, multiple-order, static Markov transition values. Transition values are given greater generality by accommodating multiple orders simultaneously, as well as the specification of transitions with the use of limited single-operator regular expressions. A complete Python implementation of this model is introduced, and high-level utilities and object interfaces are demonstrated in athenaCL (Ariza 2005). These specialized object interfaces offer flexible rhythm and parameter value generation. Additionally, the use of dynamic Markov order values is shown to offer a flexible and previously unexplored resource.

As Charles Ames states, "by no means can Markov chains be said to occupy the cutting edge of progress in automated composition…" (1989, p. 186). A Markov-based generator, further, has well-known limitations: it is "incapable of generating self-embedded structures" (Cohen 1962, p. 155) and, in general, "… is not complete enough by itself to consistently produce high quality music" (Moorer 1972, p. 111). Nonetheless, Markov chains offer a practical tool: they can be used to generate any type of value or value collection, they are discrete, they offer greater control than uniform randomness, and, at higher orders, they produce sequentially related structures. This practicality, however, is often encumbered by the parametric complexity of the traditional transition

matrix. A challenge of computer-aided algorithmic composition (CAAC) system design is the reduction of parametric complexity. This challenge can be met in part by the development of intuitive, flexible, and language-independent string notations. Such notations permit the user to supply complex specifications within a single argument, rather than supplying numerous arguments to a function or text-entry form.

2. The Markov Chain and the Markov Transition String

A Markov chain, as used here, is a technique for generating a one-dimensional series of values or symbols based on probabilities, where probabilities for each possible outcome are selected based on zero or more past symbol formations. The number of past symbols a Markov chain uses to specify a new symbol is the "order" of the Markov chain. In the case of orders greater than zero, it is useful to label these past values as a "source," and the probabilities as directing to possible "destinations" (Ames 1989, p. 175). Source formations are referred to as "transitions."

The history of Markov models is well documented in the mathematical and scientific literature (Norris 1998, Bermaud 1999). Their origins are traced to their namesake, Russian mathematician A. A. Markov (1856-1922). While some recent research has explored the Hidden Markov Model (HMM) and the Markov Chain Monte Carlo, this article focuses only on the conventional Markov "chain": a discrete-time stochastic process. Only Markov chains with finite state spaces (or a finite collection of possible symbols) are considered. While Markov chains are frequently displayed with various state diagrams or as directed graphs, are often presented in the context of random walks, and are frequently defined as regular (type 3) finite state grammars (Roads 1984, p. 14), this discussion will only consider elementary models.

A Markov chain will be defined with a new string notation. This notation encodes key and value pairs with brace-delimited values. For example, a key "x," assigned a value of 0.3, is notated as `x{0.3}`. Multiple key and value pairs can be defined in sequence without the use of spaces or commas: `x{0.3}y{0.7}`.

A Markov chain produces output based on examination of zero or more past symbols. A zeroth order Markov chain thus examines zero previous values; a third order Markov chain examines three previous values. A "transition" defines a possible past symbol formation. Thus, a second order Markov chain with two symbols (x, y) must define four transitions, or the possible past symbol formations `x:x, x:y, y:x, y:y` (where ":" separates past symbols, and symbols read from left to right as most remote to most recent). For each transition, probabilities may be assigned for the generation of a new symbol. These probabilities may be specified as weights or as unit-interval proportions. In the notation presented here, weights are defined by providing a destination symbol, an "=" sign, and a numeric value; multiple weights, separated by the "|" symbol, may be specified. Continuing the example above, the `y:x` transition may define an equal probability of producing either symbol "x" or "y" with the following notation: `x:y{x=1|y=1}`. Alternatively, the `x:x` transition could define the production of a "y" symbol nine out of ten times: `x:x{x=1|y=9}`. If a weight for a symbol is not specified within a transition, the weight is assumed to be zero.

The zero order Markov chain has one transition, that of zero previous values. The ":" character alone signifies the single transition of a zero order Markov chain. For example, a zero order transition specification for the symbols above may be defined as `:{x=3|y=4}`.

A complete Markov transition string consists of two combined parts: symbol definitions and transition weight definitions. Symbols are named with lower-case letters, and weight definitions cannot refer to undefined symbols. Transition weight definitions may be provided for any number of orders. For example, all weights defined in the previous examples may be combined with symbol definitions to demonstrate a complete Markov transition string:

Example 1. A complete Markov transition string

```
x{0.3}y{0.7}:{x=3|y=4}x:y{x=1|y=1}x:x{x=1|y=9}
```

With numerous symbols and with orders greater than zero, the number of possible transitions becomes large. To facilitate more compact transition string definitions, two features are incorporated. First, all transitions not specified are assigned an equal-weight distribution for all defined symbols. Second, transition weight definition keys may employ limited single-operator regular expressions. Three operators are permitted: "*", "-", and "|". The "*" operator matches any defined symbol: using the symbols (x, y) defined above, the transition `x:*` will match `x:x` and `x:y`. The "-" operator matches any symbol other than that specified: the transition `x:-x` will match `x:y`. The "|" operator matches any number of symbols specified: the transition `x:x|y` will match `x:x` and `x:y`.

The athenaCL system offers utility commands to both encode Markov strings (AUma) and use them as generators (AUmg). The AUma command (AthenaUtility Markov Analysis), given a maximum order and a sequence of symbols, returns the corresponding Markov string for all orders less than and equal to the order specified. For example, the following athenaCL session encodes a simple sequence of two symbols at orders zero, one, and two. Note that symbols (a and b) are automatically assigned and that the symbol sequence, under analysis, is automatically wrapped.

Example 2. Creating a Markov transition string in athenaCL with AUma

```
:: auma 2 x x x x x x y y y y y y
AthenaUtility Markov Analysis
a{x}b{y}:{a=6|b=6}a:{a=5|b=1}b:{a=1|b=5}a:a:{a=4|b=1}a:b:{b=1}b:a:{a=1}b:b:{a=1|b=4}
```

The AUmg command (AthenaUtility Markov Generator) may be used to test the generation of values from a Markov transition string with a static order. In the example below, a Markov transition string, employing limited single-operator regular expressions to define a compact second order Markov generator, is used to produce thirty values.

Example 3. Testing a Markov transition string in athenaCL with AUmg

```
:: aumg 30 2 x{a}y{b}z{c}z:-z{z=1}y:y|z{z=1|x=2}*:x{y=2|z=1}
AthenaUtility Markov Generator
b,a,c,a,b,a,b,a,b,b,a,c,b,c,a,b,b,a,c,c,a,c,a,b,b,a,b,c,a,b
```

3. The Markov Chain in the History of Algorithmic Composition

The Markov chain is one of the earliest techniques of CAAC. In the production of Experiment IV of the *Illiac Suite* (1956), Lejaren Hiller and Leonard Isaacson used zero, first, and higher order Markov chains (1959, pp. 141-148). Markov chains, with probabilities determined by analysis of excerpts from Charles Ives's *Three Places in New England*, were employed by Hiller and Robert Baker in the *Computer Cantata* (1963; Hiller and Baker 1964, p. 68; Hiller 1970, p. 54) and implemented as the ORD.n subroutine in MUSICOMP (Hiller 1969, pp. 72-73). An early and extensive exploration of computer-based Markov analysis and generation is provided by Brooks et al. (1957). Additionally, Gottfried Michael Koenig offers the related "ratio" selection method in Project Two (PR2, 1966); while not labeled as a Markov model, this generator is equivalent to a zero order Markov chain (1970). A similar utility is found in Barry Truax's POD Programs with the ratio selection mode (Buxton 1975, p. 224).

Prior to computer implementation, however, there was significant interest in generating one-dimensional sequences with Markov chains. Claude E. Shannon and Warren Weaver's 1949 text *A Mathematical Theory of Communication*, based on an earlier text by Shannon (1948) and influenced by the work of Norbert Wiener and his *Cybernetics* (1948), demonstrates the application of Markov chains for the algorithmic generation of English sentences. Shannon and Weaver, calling these generators stochastic processes, frequently suggest application to musical structures: "a system which produces a sequence of symbols (which may, of course, be letters or musical notes, say, rather than words) according to certain probabilities is called a stochastic process …" (1949, p. 11).

Following Shannon and Weaver, numerous studies employed Markov chains as musical generators. These studies were done with manual, mechanical, and primitive computer calculations, and include the analysis and generation of Western cowboy songs by Fred and Carolyn Attneave (Cohen 1962, p. 143; Quastler 1955), the "Banal Tune-Maker" of Richard C. Pinkerton (1956), and the Markov generator created in 1956 by John F. Sowa with a Geniac "Electronic Brain Kit" (Sowa 1957, 2005; Cohen 1962, p. 143). Harry Olson and Herbert Belar, in 1961, built a sophisticated electronic machine that, based on Markovian pitch and rhythm analysis of eleven Stephen Collins Foster songs, produced and synthesized new melodies (1961). The analysis data of these Foster songs has been frequently reproduced (Dodge and Jerse 1997, pp. 364-367). Additionally, the first edition of Iannis Xenakis's *Musiques Formelles* (1963) contained chapters on "Markovian Stochastic Music"; these chapters detail Xenakis's application of first order Markov chains for the selection of screens and for the generation of intensity and density values in *Analogique A* and *Analogique B* (1958-1959). These techniques were not implemented on a computer and pre-date Xenakis's Stochastic Music Program (Xenakis 1965).

Later computer-based Markov implementations are numerous and widespread. Implementations are found in Sever Tipei's MP1 (1975), David Zicarelli's Jam Factory and Joel Chadabe and Zicarelli's M (Zicarelli 1987, Chadabe 1997), the Max system, Clarence Barlow's MIDIDESK (1996 Roads 1996, p. 849), Larry Polansky and David Rosenboom's HMSL (1985) and JMSL (Didkovsky 2004), Heinrich Taube's Common Music (Taube 1989), Eduardo Reck Miranda's CAMUS 3D (McAlpine et al. 1999), Paul Berg's AC Toolbox (2003), Tim Thompson's KeyKit (Phillips 2005), and François Pachet's Continuator (2002). Additional studies and examples of Markov models are provided by W. Ross Ashby (1956), J. E. Youngblood (1958), J. E. Cohen (1962), Pierre Barbaud (1968), James Anderson Moorer (1972), Kevin Jones (1981), Ames (1989), E. Cambouropoulos (1994), D. Lyon (1995), Curtis Roads (1996, p. 878), Jon McCormack (1996), and Miranda (2000, pp. 69-72). More

recently, J. L. Triviño-Rodriguez and R. Morales-Bueno review applications of Probabilistic Suffix Automata (PSA) and related work of Assayag et al. (1999), and propose a Multiattribute Prediction Suffix Graph (MPSG) as an improvement over both Markov chains and PSA (2001), Diemo Schwarz, after a method of score following demonstrated by Orio and Déchelle (2001), employs HMMs to solve music alignment problems in concatenative synthesis (2004), David Michael Cottle demonstrates Markov models in SuperCollider 3 (2005), Miranda and Adolfo Maia Junior, introducing the Fuzzkov system, employ Markov chains with dynamic transition matrices to control grain selection (2005), and René Wooller and Andrew R. Brown discuss a technique of Markov morphing (2005). While not comprehensive, these listings demonstrate the abundance and diversity of Markov models.

4. Contemporary Markov Implementations

The Markov chain is one of the earliest techniques of CAAC (Hiller and Isaacson 1959, pp. 141-148). A few of the many contemporary Markov implementations found in CAAC systems will be examined in detail. In systems that provide modular Markov generators, input notations often directly enumerate coordinate pairs of the transition matrix as a list of two or three elements. The new string notation presented above, as well as the extended use of regular expressions, provides greater compactness and readability than these alternative notations.

Released as part of the Max library as early as 1995, the Max "prob" object provides a first order Markov generator that supports dynamic transition weights and proportional integer weight values (Dobrian 1995, pp. 318-319). Symbol and transition weight values are supplied as three-element lists with values specified in the following order: source, destination, weight. Only a single weight, to a single destination, may be defined in each list. The Max "anal" object provides a corresponding tool for first order Markov analysis, returning data lists in the appropriate format. An example, distributed with Max 4.5 ("prob.help"), demonstrates a "prob" object receiving five Markov transition weight definitions from Max message boxes. These messages, here delimited with brackets, are as follows: `[1 2 1], [2 1 1], [1 1 0], [2 2 0], [1 3 1]`.

Offering greater clarity, a single Markov transition string can encode these five messages: `a{1}b{2}c{3}a:{b=1|c=1}b:{a=1}`

Common Music (Taube 1989) offers Markov functionality with nth order Markov generation, static orders, dynamic transition weights, and unit-interval weight values. The customizable "markov" class exposes Markov functionality to the user. Common Music (CM) provides a "markov-analyze" function to generate lists of transition weights (or to configure and return a "markov" object) from a user-supplied list of data. When directly specified, transition weight definitions are provided as a space-separated Lisp list in the following form: `(source -> (destination weight))`, where destinations with equal weights may omit weight specification. In the case of higher-order rules, longer source transition patterns may be specified before the "->" symbol. The following incomplete example demonstrates the definition of numerous second order rules.

Example 4. Second order input in CM

```
(new markov
   :of '((d4 c4 -> (f4 .5) (g4 .5))
         (d4 bf4 -> bf4)
         (c4 d4 -> c4)
         (c4 c4 -> (d4 .667) (c5 0.333)))))
```

The above example can be encoded as a Markov transition string:
m{c4}n{d4}o{f4}p{g4}q{bf4}r{c5}n:m{o=1|p=1}n:q{q=1}m:n{m=1}m:m{n=2|r=1}

In CM, weight values may be supplied by dynamic pattern objects, permitting the production of Markov generators with dynamic weights. A significant feature of CM's transition specifications is support for the "*" wild-card identifier, matching any possible symbol at the designated position. A transition rule can thus be specified in the following form: (* d4 -> c4). This feature inspired the more flexible single-operator regular expression matching presented in this study.

The AC Toolbox (Berg 2003) offers Markov functionality with nth order Markov generation, static orders, static transition weights, and unit-interval weight values. Markov-based software objects are partitioned between a "transition" Generator and numerous system Tools for creating the necessary transition value table. The "transition" Generator, given this table of Markov transition probability values, allows the generation of new values according to these probabilities. There are three Tools for generating transition value tables: "Derive-Transition-Table" produces a table for a user-supplied order based on analysis of a wide variety of data objects within the AC Toolbox (such as a list, stockpile, note structure, or section); "Make-Unconditional-Table" converts a user-supplied representation of symbol weights for zeroth order transition tables; and "Make-Conditional-Table" converts a user-supplied representation of symbol weights for first and higher order transition tables. The input notation for zeroth order transitions (unconditional tables) is a Lisp list of value, weight pairs. For example:

Example 5. Zeroth order input in the AC Toolbox

```
(make-unconditional-table 'a .4 'b .4 'c .2)
```

The input notation for first and higher order transitions (conditional tables) is a Lisp list of list pairs, where each pair is a source symbol sequence and a destination weight list. This weight list follows the same format as the zero order transition above. For example:

Example 6. First order input in the AC Toolbox

```
(make-conditional-table '(a) '(b .3 c .7) '(b) '(c 1) '(c) '(a .5 b .5))
```

Demonstrating the ability to combine multiple orders in a single notation, a Markov transition string can be used to encode both examples from above:
a{a}b{b}c{c}:{a=4|b=4|c=2}a:{b=3|c=7}b:{c=1}c:{a=1|b=1}

The Markov transition string offers three advantages over these alternative representations. First, the syntax is clean and compact. Second, unique symbols are defined for referenced data. By doing so, all symbols are defined separately from weight specifications, permitting symbols to refer to complex data or long strings without obfuscating the presentation of weight assignments. Further, with all symbols explicitly defined, incomplete sets of transition weights are permitted. Third, none of the models above permit defining multiple-order weights simultaneously; with such a facility, multiple orders may be deployed from the same transition specification.

5. The Markov Transition String in athenaCL

The core athenaCL Markov implementation, as well as a complete Markov transition string parser and analysis-based string generator, is modeled as the Transition object and is defined in the Python module markov.py. This module is distributed as part of the cross platform and open source (GPL) athenaCL libATH library. Complete implementation details and object design analysis are beyond the scope of this study.

In athenaCL, ParameterObjects, as models of one-dimensional generators, offer high-level object interfaces to CAAC tools and procedures (Ariza 2005, pp. 205-207). Musical parts within athenaCL are deployed as specialized TextureModules, or multi-dimensional generators (Ariza 2005, p. 227); each Texture is configured with the assignment of numerous ParameterObjects. ParameterObjects can be supplied as arguments to other ParameterObjects, permitting complex, embedded dynamic generators.

Two ParameterObjects are provided for generating general numeric or symbolic parameter values: MarkovValue and MarkovGeneratorAnalysis. MarkovValue takes three arguments: (1) name, (2) transitionString, (3) parameterObject {order value}. The name of a ParameterObject must be provided as a first argument. A Markov transition string, of any complexity and order, is provided as a second argument. The third argument is an embedded ParameterObject to supply the order of Markov generation. As this ParameterObject may by dynamic, various alterations to Markov generation are possible. Floating-point order values are accepted, and are treated as probabilistic weightings toward surrounding integers. Thus a generated order value of 1.5 will be interpreted as an equal probabilistic weighting between 1 or 2. For each Markov value generated, these weights are evaluated and an integer order value is determined.

The MarkovGeneratorAnalysis ParameterObject takes five arguments: (1) name, (2) parameterObject {source Generator}, (3) valueCount, (4) maxAnalysisOrder, (5) parameterObject {output order value}. Rather than taking a Markov string as an argument, an embedded ParameterObject is used to produce as many values as specified by the valueCount argument, and these values are analyzed at every order up to and including the order specified by the maxAnalysisOrder argument. With this analysis data, the generator produces new values, embedding another ParameterObject to control the Markov order.

The athenaCL system features specialized ParameterObjects for rhythm generation. These generators employ Pulses, objects that specify a duration (as a ratio of an externally supplied tempo) and an accent (as an amplitude scalar between 0 and 1, or between a rest and a fully sounding event). The input notation for a Pulse is a Python list of three elements: a divisor, a multiplier, and an accent

(Ariza 2005, p. 163). Two Rhythm ParameterObjects with Markov functionality, analogous to those for general parameter values, are offered: MarkovPulse and MarkovRhythmAnalysis.

MarkovPulse takes three arguments: (1) name, (2) transitionString, (3) parameterObject {order value}. The only difference between MarkovPulse and MarkovValue is that the transition string in MarkovPulse must define symbols that refer to Pulse objects. MarkovRhythmAnalysis takes five arguments (1) name, (2) parameterObject {source Rhythm Generator}, (3) pulseCount, (4) maxAnalysisOrder, (5) parameterObject {output order value}. Similarly, the only difference between MarkovRhythmAnalysis and MarkovGeneratorAnalysis is that the analyzed ParameterObject must be a Rhythm ParameterObject.

6. Examples of Markov Generators with Dynamic Order Specifications

As cited above, musical applications and demonstrations of Markov generators are abundant in the literature. The use of dynamic and probabilistic order values, however, is uncommon, if not completely without precedent. This is in part because traditional transition matrices specify only a single order. With the Markov transition string presented here, multiple orders can be accommodated in the same representation.

For example, a sequence formed of six values spanning the unit interval may be constructed. The sequence used here is designed to present repeated oscillation followed by a narrowing of the minimum and maximum values. This sequence could be further scaled and then used for controlling amplitude, panning, or signal processing parameters. The athenaCL AUma command is used to analyze this sequence and produce a Markov transition string for orders two and lower:

Example 7. Creating a Markov transition string in athenaCL with AUma

```
:: auma 2 0 .2 .4 .6 .8 1 .8 .6 .4 .2 0 .2 .4 .6 .8 1 .8 .6 .4 .2 0 .2 .4 .6 .8 .6 .4
.2 .4 .6 .4 .6
AthenaUtility Markov Analysis
a{0}b{.2}c{.4}d{.6}e{.8}f{1}:{a=3|b=6|c=8|d=8|e=5|f=2}a:{b=3}b:{a=2|c=4}c:{b=3|d=5}d:{
a=1|c=4|e=3}e:{d=3|f=2}f:{e=2}a:b:{c=3}b:a:{b=2}b:c:{d=4}c:b:{a=2|c=1}c:d:{a=1|c=1|e=3
}d:a:{b=1}d:c:{b=3|d=1}d:e:{d=1|f=2}e:d:{c=3}e:f:{e=2}f:e:{d=2}
```

Using the MarkovValue ParameterObject, this Markov transition string, and a constant order value of two, new values are generated and are graphed in the example below. This event-domain graph, where the x axis refers to event steps and the y axis refers to generated values, is produced with the athenaCL TPmap (TextureParameter Map) command. Note that upward and downward oscillation is retained, with a slightly higher frequency of oscillation between values in the middle of the range.

Example 8. MarkovValue generation at order 2
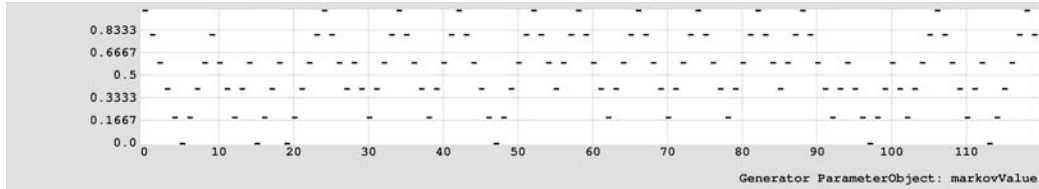
In the next example, the same transition string is used, while the order is set to a constant value of one. Note that oscillation gets "stuck" within value pairs, demonstrating that only one previous value is taken into the context of generating new values.

**Example 9. MarkovValue generation at order 1**
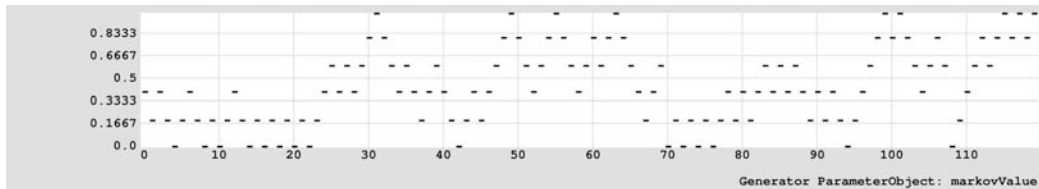
In the example below, the same transition string is used while the order is set to a constant value of zero. No longer is oscillation clearly visible: instead, a distribution of values proportional to their analyzed frequency is created.

Example 10. MarkovValue generation at order 0
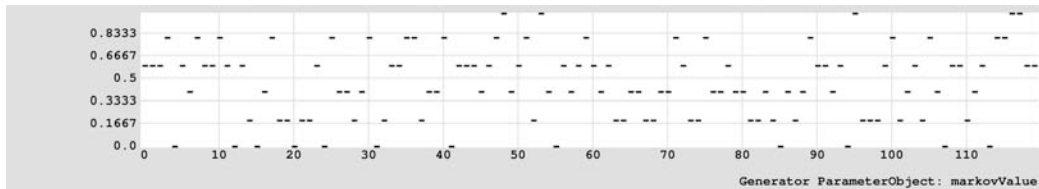
In the next example, the order parameter is linearly increased from 0 to 2 over the span of 120 events. As a clear movement between different behaviors is apparent, this example demonstrates, even within the small range of 120 events, the utility of employing dynamic order values.

Example 11. MarkovValue generation with a dynamic order

Generator ParameterObject: markovValue
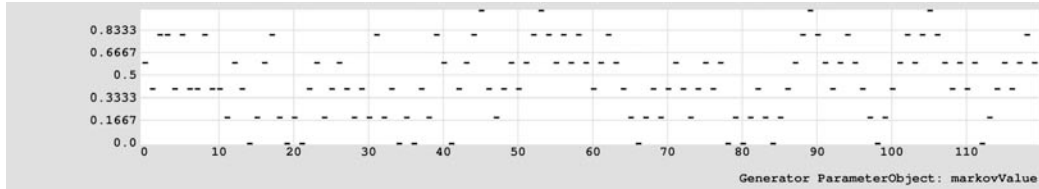
```
markovValue,
a{0}b{.2}c{.4}d{.6}e{.8}f{1}:{a=3|b=6|c=8|d=8|e=5|f=2}a:{b=3}b:{a=2|c=4}c:{b=3|d=5}d:{
a=1|c=4|e=3}e:{d=3|f=2}f:{e=2}a:b:{c=3}b:a:{b=2}b:c:{d=4}c:b:{a=2|c=1}c:d:{a=1|c=1|e=3
}d:a:{b=1}d:c:{b=3|d=1}d:e:{d=1|f=2}e:d:{c=3}e:f:{e=2}f:e:{d=2}, (breakPointLinear,
event, single, ((0,0),(119,2)))
```

A final example uses the MarkovGeneratorAnalysis ParameterObject to demonstrate the use of dynamic order values with a Markov transition string generated by analysis of an embedded ParameterObject. In this example second order and lower Markov analysis is performed on thirty values from a sine wave with a period of thirty events. The resulting transition data is used to generate new values, with the output Markov generator order determined by an embedded MarkovValue ParameterObject. This ParameterObject, using a zero order Markov chain, produces order values weighted toward second order, with less frequent first and zero order values. The values produced favor continuous segments of a re-generated sine wave, with interruptions of first and zero order re-generated fragments.

Example 12. MarkovGeneratorAnalysis generation with a MarkovValue-controlled dynamic order

Generator ParameterObject: markovGeneratorAnalysis
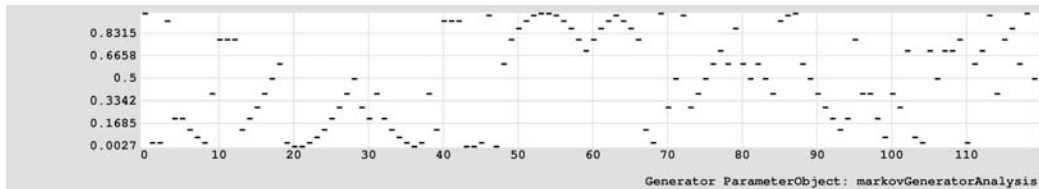
```
markovGeneratorAnalysis, (waveSine, event, 30, 0, (constant, 0), (constant, 1)), 30,
2, (markovValue, a{0}b{1}c{2}:{a=3|b=7|c=12}, (constant, 0))
```

7. Conclusion

Markov chains offer a flexible CAAC tool for the probabilistic generation of parameter values. As a tool with a long history, opportunities for innovation are limited. As this study demonstrates, the development of more flexible parametric interfaces through powerful notations offers an avenue of exploration.

8. Acknowledgments

References

Ames, C. 1989. "The Markov Process as a Compositional Model: A Survey and Tutorial." *Leonardo* 22(2): 175-187.

Ariza, C. 2005. *An Open Design for Computer-Aided Algorithmic Music Composition: athenaCL*. Ph.D. Dissertation, New York University.

Ashby, R. 1956. *An Introduction to Cybernetics*. London: Chapman & Hall Ltd.

Assayag, G. and S. Dubnov, O. Delerue. 1999. "Guessing the Composer's Mind: Applying Universal Prediction to Musical Style." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 496-499.

Barbaud, P. 1968. *La musique discipline scientifique*. Paris: Dunod.

Berg, P. 2003. *Using the AC Toolbox*. Den Haag: Institute of Sonology, Royal Conservatory.

Bremaud, P. 1999. *Markov Chains*. London: Springer.

Brooks, F. P. and A. Hopkins, P. Neumann, W. V. Wright. 1957. "An Experiment in Musical Composition." *IRE Transcripts on Electronic Computers* 6: 175-182.

Buxton, W. 1975. *Manual for the POD Programs*. Utrecht: Institute of Sonology, University of Utrecht.

Cambouropoulos, E. 1994. "Markov Chains As an Aid to Computer Assisted Composition." *Musical Praxis* 1(1): 41-52.

Chadabe, J. 1997. *Electric Sound: The Past and Promise of Electronic Music*. New Jersey: Prentice-Hall.

Cohen, J. E. 1962. "Information Theory and Music." *Behavioral Science* 7(2): 137-163.

Cottle, D. M. 2005. "Computer Music with examples in SuperCollider 3."

Didkovsky, N. 2004. "Java Music Specification Language, v103 update." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 742-745.

Dobrian, C. 1995. *MAX Reference*. Mountain View: Opcode Systems.

Dodge, C. and T. A. Jerse. 1997. *Computer Music: Synthesis, Composition, and Performance*. New York: Shirmer Books.

Hiller, L. 1969. "Some Compositional Techniques Involving the Use of Computers." In *Music by Computers*. H. von Foerster and J. W. Beauchamp, eds. New York: John Wiley & Sons, Inc. 71-83.

Hiller, L. 1970. "Music Composed with Computers: An Historical Survey." In *The Computer and Music*. H. B. Lincoln, ed. Ithaca: Cornell University Press. 42-96.

Hiller, L. and R. Baker. 1964. "Computer Cantata: A Study in Compositional Method." *Perspectives of New Music* 3(1): 62-90.

Hiller, L. and L. Isaacson. 1959. *Experimental Music*. New York: McGraw-Hill.

Jones, K. 1981. "Compositional Applications of Stochastic Processes." *Computer Music Journal* 5(2): 45-61.

Koenig, G. M. 1970. "Project Two - A Programme for Musical Composition." In *Electronic Music Report*. Utrecht: Institute of Sonology. 3.

Lyon, D. 1995. "Using Stochastic Petri Nets for Real-Time Nth-Order Stochastic Composition." *Computer Music Journal* 19(4): 13-22.

McAlpine, K. and E. Miranda, S. Hoggar. 1999. "Making Music with Algorithms: A Case-Study." *Computer Music Journal* 23(2): 19-30.

McCormack, J. 1996. "Grammar Based Music Composition." In *Complex Systems 96: From Local Interactions to Global Phenomena*. R. Stocker, ed. Amsterdam: ISO Press.

Miranda, E. R. 2000. *Composing Music With Computers*. Burlington: Focal Press.

Miranda, E. R. and A. M. Junior. 2005. "Granular Synthesis of Sounds Through Markov Chains with Fuzzy Control." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 193-196.

Moorer, J. 1972. "Music and Computer Composition." *Communications of the ACM* 15(2): 104-113.

Norris, J. R. 1998. *Markov Chains*. Cambridge: Cambridge University Press.

Olson, H. F. and H. Belar. 1961. "Aid to Music Composition Employing a Random Probability System." *Journal of the Acoustical Society of America* 33(9): 1163-1170.

Orio, N. and F. Déchelle. 2001. "Score Following Using Spectral Analysis and Hidden Markov Models." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 125-129.

Pachet, F. 2002. "The Continuator: Musical Interaction with Style." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 211-218.

Phillips, D. 2005. "At the Sounding Edge: Introducing KeyKit." LINUX Journal. Internet: http://www.linuxjournal.com/article/8153

Pinkerton, R. C. 1956. "Information Theory and Melody." *Scientific American* 194(2): 77-86.

Polansky, L. and D. Rosenboom. 1985. "HMSL." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 243-250.

Quastler, H. 1955. "Discussion, following Mathematical theory of word formation, by W. Fucks." In *Information Theory: Third London Symposium*. E. C. Cherry, ed. New York: Academic Press. 168.

Roads, C. 1984. "An Overview of Music Representations." In *Musical Grammars and Computer Analysis*. Firenze: Leo S. Olschki. 7-37.

Roads, C. 1996. *The Computer Music Tutorial*. Cambridge: MIT Press.

Schwarz, D. 2004. *Data-Driven Concatenative Sound Synthesis*. Ph.D. Thesis, Ircam, University of Paris 6.

Shannon, C. E. 1948. "A Mathematical Theory of Communication." *Bell Systems Technical Journal* 27: 379-423, 623-656.

Shannon, C. E. and W. Weaver. 1949. *A Mathematical Theory of Communication*. Urbana: University of Illinois Press.

Sowa, J. F. 1957. "A machine to compose music." In *Geniac Manual*. New York: Oliver Garfield Company.

Sowa, J. F. 2005. Personal correspondence. 25 July 2005.

Taube, H. 1989. "Common Music: A Compositional Language in Common Lisp and CLOS." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 316-319.

Tipei, S. 1975. "MP1 — a Computer Program for Music Composition." In *Proceedings of the Second Annual Music Computation Conference*. J. Beauchamp and J. Melby, eds. Urbana, Illinois: Office of Continuing Education and Public Service in Music. 68-82.

Trivino-Rodriguez, J. L. and R. Morales-Bueno. 2001. "Using Multiattribute Prediction Suffix Graphs to Predict and Generate Music." *Computer Music Journal* 25(3): 62-79.

Wiener, N. 1948. *Cybernetics*. Cambridge: MIT Press.

Wooller, R. and A. R. Brown. 2005. "Investigating morphing algorithms for generative music." *Third Iteration: Third International Conference on Generative Systems in the Electronic Arts*.

Xenakis, I. 1963. *Musiques Formelles*. Paris: Editions Richard-Masse.

Xenakis, I. 1965. "Free Stochastic Music from the Computer. Programme of Stochastic music in Fortran." *Gravesaner Blätter* 26.

Youngblood, J. E. 1958. "Style as Information." *Journal of Music Theory* 2(24).

Zicarelli, D. 1987. "M and Jam Factory." *Computer Music Journal* 11(4): 13-29.

21M.380 Music and Technology: Algorithmic and Generative Music
Spring 2010