# Music Generation with Markov Models

**Walter Schulze and Brink van der Merwe**
*Stellenbosch University*

**By using music written in a certain style as training data, parameters can be calculated for Markov chains and hidden Markov models to capture the musical style of the training data as mathematical models.**

In this article, we investigate whether a computer can compose music that is just as pleasurable to the average human listener as music composed by a human with moderate experience. In our system, music is composed by separating the composition process into steps, such as chord progression, melodic curve, cadence, and so forth, as proposed by Högberg.[1] For each of these steps, we calculate parameters for probabilistic automata from music data, then use these probabilistic automata to generate new compositions.

Our system, SuperWillow, is a modification and extension of the composition system Willow.[1] Willow is a rule-based, music-composition system that works on the premise that the composition process can be modeled by tree operations on ranked trees used as music representation. In SuperWillow, symbolic music (more specifically, MusicXML, see http://www.musicxml.org/) is used to represent the partial composed music obtained after each of the composition steps. Also, Extensible Stylesheet Language and Document Object Model, instead of ranked top-down tree transducers, as in the case of Willow, are used to perform each of the composition steps.

Just as an artist listens to other artists and styles that influence his or her compositions, our system learns from input compositions and generates output compositions in the same style. This is similar in spirit to the hidden Markov model (HMM) approach used for classification of folk music.[2] (See the ''Related Work'' sidebar for a discussion of other projects in this area.) Two-voice harmonization can be achieved by composing chords for a given melody,[3,4] or by composing the melody for a given sequence of chords, as in Willow. We use HMMs to harmonize two-instrument music and prediction-suffix trees to model the melody and rhythm. We also divide the composition process into several steps to keep the state space manageable.

We evaluated our music-generation system (available for download at http://superwillow.sourceforge.net) with a partial Turing test. In a survey, we asked respondents to identify the human composition from two compositions, one composed by a human and one generated by our system. We obtained promising results: 38 percent of respondents in the survey incorrectly attributed music composed by our computer system to a human composer.

## Implementation

Building probabilistic automata for our music system requires filtering, extraction, and analyses of the music data. A filter is used to remove music pieces that SuperWillow cannot analyze, such as a piece with a time-signature change. During extraction, the music data is transposed to the key of C while preserving the scale type. Then a list of classified chords and note durations are obtained from the music data. The analyzer component of SuperWillow creates analysis objects that include Markov chains (MCs) and HMMs. After analyzing the music data, the analysis objects are grouped into styles. By using these analysis objects, the system generates or imitates compositions in a given style.

### Automata and Markov models

MCs are used in SuperWillow to model chord duration, chord progression, and rhythm progression. MCs are used to model a sequence of events by using states and transition probabilities between states. An MC adheres to the first order Markov assumption, which states that

$$P(q_t \mid q_{t-1}, q_{t-2}, \ldots, q_1) = P(q_t \mid q_{t-1}),$$

where $q_1, \ldots, q_t$ is a set of states. Thus, in an MC, the probability of moving to state $q_t$, at time t, depends only on the state, at time $(t - 1)$. We use homogeneous MCs that have time-independent transition probabilities. Thus the probability of moving to a

1070-986X/11/$26.00 © 2011 IEEE

## Related Work

Much research in this area has focused on capturing elements of music composition by prediction suffix trees.[1-3] For example, Pachet used predication suffix trees in the Continuator,[4] an interactive MIDI imitator. Prediction suffix trees are effective at style modeling, but might describe compositions too closely in terms of producing compositions that share long identical sequences of notes with the music being modeled. Also, the prediction suffix tree method is limited by the fact that it can only compose music consisting of one voice, in particular when musical events are not synchronous.

Hidden Markov models (HMMs), on the other hand, have been used successfully to model the relation between pitches played by different voices, for instance when harmonizing chorales.[5] Simon, Morris, and Basu[6] used HMMs in MySong, commercially marketed by Microsoft under the name Songsmith, which is a system where users provide a melody by singing into a microphone, and where the system then composes complementing chords. These HMM implementations generate pitches for notes with fixed durations and do not provide any real rhythm generation.

### References

1. D. Conklin and C. Anagnostopoulou, ''Representation and Discovery of Multiple Viewpoint Patterns,'' *Proc. Int'l Computer Music Conf.,* Int'l Computer Music Assoc., 2001, pp. 479-485.
2. S. Dubnov et al., ''Using Machine-Learning Methods for Musical Style Modeling,'' *Computer,* vol. 36, no. 10, 2003, pp. 73-80.
3. J. Triviño-Rodriguez and R. Morales-Bueno, ''Using Multiattribute Prediction Suffix Graphs to Predict and Generate Music,'' *Computer Music,* vol. 25, no. 3, 2001, pp. 62-79.
4. F. Pachet, ''The Continuator: Musical Interaction with Style,'' *New Music Research,* vol. 32, no. 3, 2003, pp. 333-341.
5. A. Moray, and C.K.I. Williams, ''Harmonising Chorales by Probabilistic Inference,'' *Advances in Neural Information Processing Systems,* vol. 17, L.K. Saul, Y. Weiss, and L. Bottou, eds., MIT Press, 2005, pp. 25-32.
6. I. Simon, D. Morris, and S. Basu, ''Mysong: Automatic Accompaniment Generation for Vocal Melodies,'' *Proc. SIGCHI Conf. Human Factors in Computing Systems,* ACM Press, 2008, pp. 725-734.

next state, at any time, only depends on the current state. The transition probabilities of MCs are estimated by calculating the likelihood estimate using frequency or empirical counts. In contrast to MCs, higher-order MCs have a memory length larger than one. More precisely, an $L$-th order MC operates under the assumption that

$$P(q_t \mid q_{t-1}, \ldots, q_1) = P(q_t \mid q_{t-1}, \ldots, q_{\max(t-L,1)}).$$

This results in multiple transition probabilities between states $q_{t-1}$ and $q_t$. The complication introduced by higher order MCs can be removed by increasing the number of states and adding a history to the states. To transform an $L$-th order MC to a first-order MC, the state space $Q$ is replaced by $\cup_{i=1}^{L} Q^i$, where $Q^i$ is the set of state sequences of length $i$. By doing this, it is possible to convert a higher-order MC to an equivalent first-order MC.

Prediction suffix automatons (PSAs)[5,6] are equivalent to mixed-order MCs. Mixed-order MCs allow transitions of various memory lengths, whereas all transitions of higher-order MCs have the same memory length. This flexibility of memory lengths avoids the exponential growth associated with higher-order MCs. PSAs represent memory using state labels in a fashion similar to first-order MCs obtained from translating higher-order MCs to their equivalent first-order MCs. A PSA is a four-tuple $\langle \Sigma, Q, \tau, p \rangle$ where:

- $\Sigma$ is the finite input alphabet;
- $Q \subset \Sigma^*$, the state space, is a finite set of finite-length strings with $\lambda \in Q$;
- $\tau : Q \times \Sigma \to Q$ is the state transition function; and
- $p : Q \times \Sigma \to [0, 1]$ is the next symbol-probability distribution.

The following constraints must be satisfied:

- $\Sigma_{\sigma \in \Sigma} p(q, \sigma) = 1$ for all $q \in Q$;
- the start state $q_0$ is the empty string $\lambda$; and
- for all $q \in Q$ and $\sigma \in \Sigma$, $\tau(q, \sigma)$ is equal to the longest suffix of $q\sigma$ that is in $Q$.

Training an $L$-PSA (a PSA with a maximum memory length of $L$) is achieved by using the

*Figure 1. A prediction suffix automaton.*

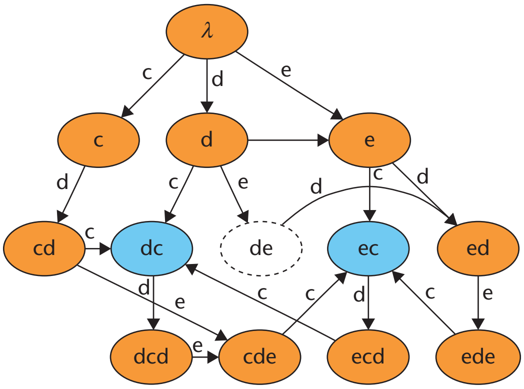

*Table 1. Training data for the hidden Markov models in Figure 2 where x is the observation sequences and s is the hidden state sequences.*

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | I | I | I | V | V | V | I | I | V | V |
| $s_1$ | c | d | e | c | d | c | d | e | c | d |
| $x_2$ | I | I | I | I | V | V | V | V | I | I |
| $s_2$ | d | e | d | e | c | d | c | d | e | c |

LearnPSA algorithm.[5] The LearnPSA algorithm produces a prediction suffix tree which is then converted to a PSA. The algorithm's design was motivated by the probably approximately correct (PAC) learning model. This algorithm finds all the strings with a statistical significance (also called motifs), given certain input parameters for the training sequences. These parameters are the maximum length of a sequence, the minimum probability of a sequence occurring in the training data, and a factor by which the conditional probability of the sequence must be bigger than that of its suffixes.

Figure 1 shows the resulting PSA given the two training sequences: (c, d, e, c, d, c, d, e, c, d) and (d, e, d, e, c, d, c, d, e, c). Note that the node de is only added after the completion of the LearnPSA algorithm. Although de is not statistically significant, it is needed to complete the constructed tree because it is the parent of cde and ede. The nodes dc and ec are added to the PSA to make nodes dcd and ecd reachable. The suffixes ec and dc were not added to the original tree because they have the same probability distribution as their parent c. The parent c is always followed by d, whether its predecessor is e or d.

Converting a prediction suffix tree to a PSA is achieved by adding all missing prefixes to the state space and by constructing the transition function $\tau$. Each state needs all its prefixes to allow the state to be reachable in the PSA. These prefixes inherit their transition probabilities from their longest proper suffix in the state space. Constructing $\tau$ involves finding the destination state for each source state and transition symbol. The destination state of a transition is the longest suffix of the string obtained by concatenating the source state and the transition symbol, which is also in the state space.

HMMs are used to model the relation between a hidden and an observed sequence. A discrete HMM is an MC with a discrete probability distribution at each state. These discrete probability distributions define probabilities of emitting a specific alphabet symbol in a given hidden state. Thus the states of the MC are the hidden states. A discrete HMM[7] is a five-tuple $\langle \Sigma, Q, a, b, \pi \rangle$ where:

- $\Sigma$ is a finite alphabet of visible symbols;
- $Q$ is a finite set of hidden states;
- $a : Q \times Q \rightarrow [0, 1]$ is a mapping defining the probability of transitions between hidden states;
- $b : Q \times \Sigma \rightarrow [0, 1]$ is a mapping defining the emission probability of each visible symbol at a given hidden state, also called a confusion matrix; and
- $\pi : Q \rightarrow [0, 1]$ is a mapping that defines the initial probability of the hidden states.

The following constraints must be satisfied:

- for all $q_i \in Q, \Sigma_{q_j \in Q} a(q_i, q_j) = 1$;
- for all $q \in Q, \Sigma_{\varsigma \in \Sigma} b(q, \varsigma) = 1$; and
- $\Sigma_{q \in Q} \pi(q) = 1$.

In the case where the hidden state sequence and observation sequence are available as training data, empirical counts can be used to calculate the parameters of the model. Table 1 gives the hidden and observed sequences used to calculate the parameters of the HMM in Figure 2. The hidden state sequence can be used to

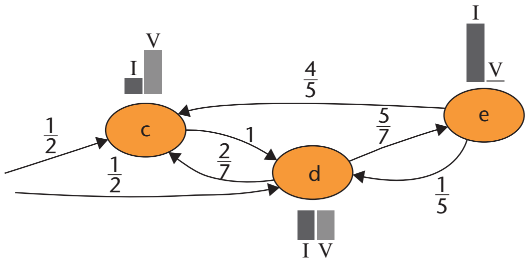

***Figure 2. A hidden Markov model with the discrete probabilistic distribution at each state displayed as a histogram.***

determine the transition probabilities of the underlying MC. Thus:

$$a = \begin{pmatrix} 0 & 1 & 0 \\ \frac{2}{7} & 0 & \frac{5}{7} \\ \frac{4}{5} & \frac{1}{5} & 0 \end{pmatrix}$$

and

$$\pi = \left(\frac{1}{2}, \frac{1}{2}, 0\right).$$

The confusion matrix, $b$, is also trained using counts. For all $q \in Q$ and $\sigma \in \Sigma$, we have that:

$$b(q, \sigma) = \frac{\#(s_1^t = q \quad and \quad \chi_1^t = \sigma)}{\#(s_1^t = q)}$$

Thus:

$$b = \begin{pmatrix} b(c, I) & b(c, V) \\ b(d, I) & b(d, V) \\ b(e, I) & b(e, V) \end{pmatrix} = \begin{pmatrix} \frac{2}{7} & \frac{5}{7} \\ 0.5 & 0.5 \\ 1 & 0 \end{pmatrix}$$

First-order HMMs consist of a first-order MC with a probability distribution associated with each state of the MC. More generally, a mixed-order HMM is a mixed order MC with a probability distribution associated with each state of the MC. Instead of using a mixed-order MC, as the base of the mixed-order HMM, we use a PSA because PSAs are equivalent to mixed-order MCs. Each state of the PSA used in a given mixed-order HMM is associated with a probability distribution defining its emission probabilities. This probability distribution is the same for all states in the PSA that have the same last symbol in their respective state labels.

The mixed-order nature of a PSA is kept in the state labels, instead of the transitions as is the case for mixed-order MCs. This property makes the PSA's transitions first order. Figure 3 shows the PSA in Figure 1 with associated probability distributions. The distributions $f_c$, $f_d$, and, $f_e$ are represented by the rows of the confusion matrix.

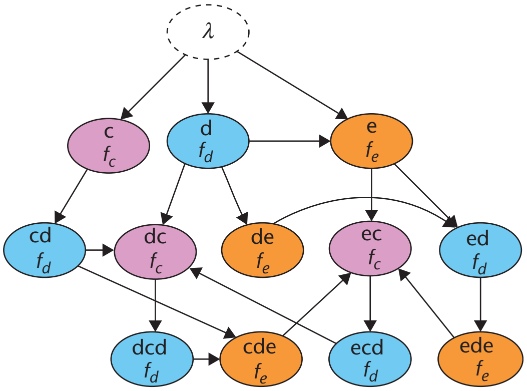

***Figure 3. A mixed order Markov model.***

Final or acceptance states in an automaton require the automaton to be in one of these states after the processing of a string, to accept the given string. The automata described before did not include final states because they only modeled strings of a fixed length, implying that their distributions were normalized over strings with the same length. The introduction of final states normalizes the probability distribution of generated strings over all lengths.

### Music analysis

After extracting the relevant information from each MusicXML file used as data, the extracted information is analyzed to retrieve the parameters required by the XML operations. The tempo, scale, and time signature are retrieved straightforwardly. The chord duration, chord progression, and rhythm progression, are all represented by first-, higher-, or mixed-order MCs. The melodic arc is described by a first-, higher-, or mixed-order HMM. The chords in the chord progression and melodic arc are represented by roman numerals. Generating chords requires the roman numerals to represent actual chords. These chords are extracted from the music data by the accompaniment analysis. The user specifies which voice is the lead and which is the accompaniment.

The chord duration analysis is applied to the accompaniment voice. The parameters of the MC, which is used to model the chord durations, are calculated by using frequency counts. The training sets are simply lists of durations. These durations are calculated by grouping

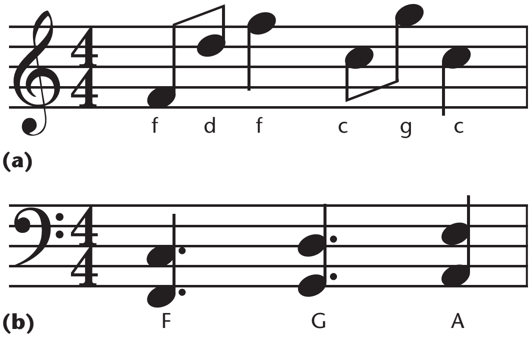

*Figure 4. Music in the scale of C major used as training data: (a) lead and (b) accompaniment.*

and summing durations of chords that are the same and that follow each other. The chord progression analysis is applied to the accompaniment voice. The training sets, which are lists of roman numerals, are used to calculate the parameters of an MC by using frequency counts. These roman numerals represent the place of each root key in the scale.

The accompaniment analysis stores each unique chord as its root key or associated Roman numeral and type. Roman numerals are generated as the chord progression. These roman numerals or root keys are later assigned a type on the basis of the data stored by the accompaniment analysis.

The rhythm progression analysis is applied to the accompaniment and lead voice. Again, the parameters of the MC used to model the rhythm progression, are calculated by using frequency counts. The rests and unknown chords are excluded from the training data. This exclusion possibly divides the training set into multiple training sets, but only the first and last set include a start and final state because this is where they would have been if the training set weren't divided. The alphabet of the MC labels the states by the duration associated with each note.

The melodic arc analysis calculates the parameters of an HMM of first, higher, or mixed order by using empirical counts to model the melodic arc. The sequence is described by Roman numerals representing the accompaniment, while the hidden sequence is calculated from the melody. The hidden alphabet is a combination of the index of the melody note in the scale and its relation to the previous note, although it is not really hidden because the melody notes are available. The relation to the previous note, also known as the contour, is represented by a plus, minus, or equal sign, depending on whether the current note is higher, lower, or equal to the previous note. The hidden sequence of the music piece in Figure 4 is {=4, +2, +4, −1, +3, −1} because f is the fourth pitch in the C major scale.

Each melody note is matched to an accompaniment chord in the observed sequence. The chord that the accompaniment voice was playing, when the melody note was sounded first, is chosen to be the matching chord. The observed sequence of the music piece in Figure 4 is {IV, IV, IV, V, V, VI}. Once all music pieces are analyzed, they are combined into analysis objects representing the style of an artist or genre. The grouping of the music pieces, used as data, is left to the end user.

### Music generation

After analyzing the compositions, the analysis objects are used to generate new compositions or to imitate the music that was used as training data. Generation is not achieved by calculating the most probable sequence of notes, but rather by sampling from the distributions obtained from the analysis. First, a generation style is chosen randomly or specified by the user. Then the tempo, time signature, scale, and respective instruments are randomly chosen from those found in the selected style.

The chord duration and rhythm progression MCs are used to generate a note value sequence that sums to a specified duration. This process is achieved by extending the state space. In addition to the original PSA state space, it includes a duration left value, which is the duration specified minus the duration of the summed note values in the generated sequence. The sequence is generated so at the end of the generated sequence, the duration left value is equal to zero.

Carmel is a finite-state transducer package. Transducers, which are generalized automata with both input and output symbols on transitions,[8] operate by producing output symbols as input symbols are consumed. Carmel is used to sample from the distributions created by analysis of the compositions used as data. Rhythm generation, as described, uses a probabilistic automaton which is piped through Carmel. Chord progression uses a probabilistic transducer to transform an input string of the length of the required chord progression.

The melodic arc uses noisy channel decoding as described in the tutorial on Carmel's website (see http://www.isi.edu/licensed-sw/carmel/).

The hidden state transitions of the HMM are represented by a probabilistic automata, while the confusion matrix is described by a probabilistic transducer. We can use Carmel for some of our calculations because HMMs can be converted to equivalent probabilistic transducers.

SuperWillow allows the user to choose the order and mixed nature of the MCs used, as well as the length of the composition to be generated. As is usually the case when using higher-order Markov models, sparseness of training data could be problematic. We determined experimentally that if all reasonable note sequences are given nonzero probability by smoothing, the generated music is not necessarily of a higher quality.

## Evaluation

Style imitation can be considered to be an artificial-intelligence process, and thus a partial Turing test can be applied.[3,4] We conducted a survey in which respondents were asked to select the composition that was composed by a human composer, with one being composed by a human and the other by a computer. We also asked where the respondent ranked three compositions, from favorite to least favorite. The three compositions consisted of a composition composed by a human and two system-generated compositions, one being composed using higher- and the other mixed-order MCs and HMMs. To ensure fairness, the human composers were given the same constraints as the computer system.

### Survey compositions

Our evaluation consisted of two surveys each representing a different style. The two styles used music from a different human composer (or composers) as data. The first style, Burger, consisted of four compositions by Eduard Burger, who has 10 years of guitar playing experience and eight years of composition experience, preferring the style of metal. The second style, melted, consisted of two compositions by John Charles Dalton, who has music theory grade eight from the Royal School of Music, flute grade eight, and classical guitar grade six, and one composition each from affron5 and MindAtrophy, who are two users of the online music composition software Noteflight (see http://www.noteflight.com/), with unknown music experience. Using such small data sets to infer music styles could be seen as overfitting of the music data, but it was not possible to amass larger data sets due to time constraints and limits placed on the music accepted as data by SuperWillow.

### Composition constraints

Human composers were asked to compose pieces with two instruments, one playing chords and the other a melody, of more or less eight measures in length. This does not include the Noteflight users because their compositions were selected on the basis of these criteria. The compositions were kept short because music generated by Markov models don't have an overhanging structure. The lack of overhanging structure, including phrases and their ordering, is more noticeable for longer compositions. The constraints placed on the compositions created by the human composers were as follows:

- the time signature should be constant throughout a composition;
- the notes of a chord should be sounded at the same time and not arpeggiated;
- both instruments should play without any rests;
- the compositions should conclude with a cadence;
- notes should not sound from one measure to another; and
- only note durations from a whole to a sixteenth should be used, and durations such as triplets should not be used.

In addition, all compositions were in C major with four quarter beats per measure.

Several compositions were generated in each style, from which three compositions were chosen. One of the selected compositions, generated using mixed-order MCs and HMMs, was used with one of the human compositions from which it was generated, for the purpose of conducting the Turing test. The ranking question consisted of two selected compositions of mixed and higher order and another human composition from which they were generated. The generated compositions were placed against the compositions used as input

*Table 2. Results of survey question 1.*

| Style | Identified incorrectly | Identified correctly |
|---|---|---|
| Burger | #54 = 44% | #70 = 56% |
| Melted | #47 = 34% | #92 = 66% |
| Total | #101 = 38% | #162 = 62% |

because they represent the same style of music. In this way, style bias was eliminated.

The compositions generated for the Burger style consisted of five compositions for each (higher and mixed order) Markov model of memory length up to 10. Unfortunately, for the melted style we could only generate higher order up to a memory length of two and mixed order up to a memory length of five, due to sparseness of the music data.

### Results

The survey was hosted on a website and completed by 263 respondents, consisting mostly of staff and students from Stellenbosch University and their respective friends and colleagues. The first survey question was set up as a partial Turing test to discover whether a human could distinguish between a composition generated by our system and a composition composed by a human. Table 2 shows that 38 percent of respondents made an incorrect choice.

The next survey question asked the respondents to rank three given compositions in order of preference. The results are given in Figure 5a. As shown in Figure 5b, 59 percent of respondents indicated their preference for one or both of the two given computer compositions over a human composition. A thorough investigation into the similarity between generated music and training data is still required. A detailed discussion of the evaluation results can be found elsewhere.[9]

## Conclusion

The survey data showed that human compositions are still preferred, but also that 72 percent of respondents either could not distinguish between a human and computer composition, or preferred a computer composition over a human composition. Dividing the music-generation process into several operations reduces the state space and processing power needed for each generation step. Unfortunately, the fact that the generation steps do not take each other into account results in failed composition attempts, for example when the rhythm progression is unable to generate note values that fit into the generated chord durations.

We established that HMMs are well suited to model the relationship between a melody and accompanying chords. Unfortunately the benefit of mixed order over higher order was not indicated by the survey results. As expected, we did find that mixed order was able to produce compositions of a much higher memory length, compared to higher order. At this point, SuperWillow lacks a proper analysis of

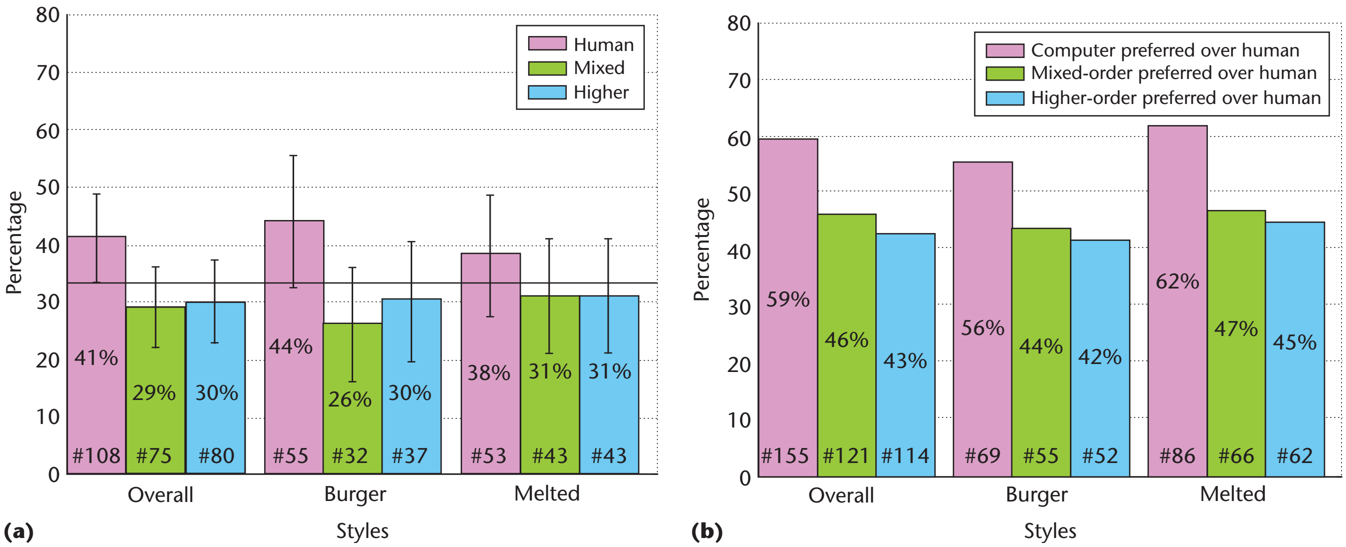

*Figure 5. Histograms representing the percentage of respondents that preferred (a) a specific composition and (b) at least one of the computer compositions over the human composition.*

rhythm and rests, can only generate compositions with two voices, and places too many restrictions on the training data. A future system will address these issues. **MM**

## References

1. J. Högberg, *Wind in the Willows,* tech. report UMINF 05.13, Umeå Univ., 2005; http://www.cs.umu.se/~johanna/willow/.
2. W. Chai and B. Vercoe, ''Folk Music Classification Using Hidden Markov Models,'' *Proc. Int'l Conf. Artificial Intelligence,* 2001.
3. J. Triviño-Rodriguez and R. Morales-Bueno, ''Using Multiattribute Prediction Suffix Graphs to Predict and Generate Music,'' *Computer Music,* vol. 25, no. 3, 2001, pp. 62-79.
4. F. Pachet, ''The Continuator: Musical Interaction with Style,'' *New Music Research,* vol. 32, no. 3, 2003, pp. 333-341.
5. D. Ron, Y. Singer, and N. Tishby, ''The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length,'' *Machine Learning,* vol. 25, no. 2, 1996, pp. 117-149.
6. L. Schwardt, *Efficient Mixed-Order Hidden Markov Model Inference,* doctoral dissertation, Univ. of Stellenbosch, 2007.
7. P. Dupont, F. Denis, and Y. Esposito, ''Links between Probabilistic Automata and Hidden Markov Models: Probability Distributions, Learning Models and Induction Algorithms,'' *Pattern Recognition,* vol. 38, no. 9, 2005, pp. 1349-1371.
8. D. Jurafsky and J.H. Martin, *Speech and Language Processing,* 2nd ed., Prentice Hall, 2008.
9. W. Schulze, ''A Formal Language Theory Approach to Music Generation,'' master's thesis, Univ. of Stellenbosch, 2009.

**Walter Schulze** is a graduate student in computer science at Stellenbosch University. His research interest is in machine learning. Schulze has an MS in computer science from Stellenbosch University. Contact him at awalterschulze@gmail.com.

**Brink van der Merwe** is a professor of computer science at Stellenbosch University. His research interest is in automata theory. Van der Merwe has a PhD in mathematics from Texas A&M University. Contact him at abvdm@cs.sun.ac.za.