Chapter 8

Algorithms and other topics

Lecture notes 2005

8.1 Markov Models and Hidden Markov Models

Andrei Andreevich Markov first introduced his mathematical model of dependence, now known as Markov chains, in 1907. A Markov model (or Markov chain) is a mathematical model used to represent the tendency of one event to follow another event, or even to follow an entire sequence of events. Markov chains are matrices comprised of probabilities that reflect the dependence of one or more events on previous events. Markov first applied his modeling technique to determine tendencies found in Russian spelling. Since then, Markov chains have been used as a modeling technique for a wide variety of applications ranging from weather systems to baseball games.

Statistical methods of Markov source or hidden Markov modeling (HMM) have become increasingly popular in the last several years. The models are very rich in mathematical structure and hence can form a basis for use in a wide range of applications. Moreover the models, when applied properly, work very well in practice for several important applications.

8.1.1 Markov Models or Markov chains

Markov models are very useful to represent families of sequences with certain specific statistical properties. To explain the idea consider a simple 3 state model of the weather. We assume that once a day, the weather is observed as being one of the following: rain (state 1); cloudy (state 2); sunny (state 3).

If we examine a sequence of observation during a month, the state rain appears a few times, and it can be followed by rain, cloud or sun. Given a long sequence of observations, we can count the number of times the state rain is followed by, say, a cloudy state. From this we can estimate the probability that a rain is followed by a cloudy state. If this probability is 0.3 for example, we indicate it as shown in Figure 8.1. The figure also shows examples of probabilities for every state to transition to other states, including itself. The first row of the matrix A

$$A = \{a_{i,j}\} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix} \tag{8.1}$$

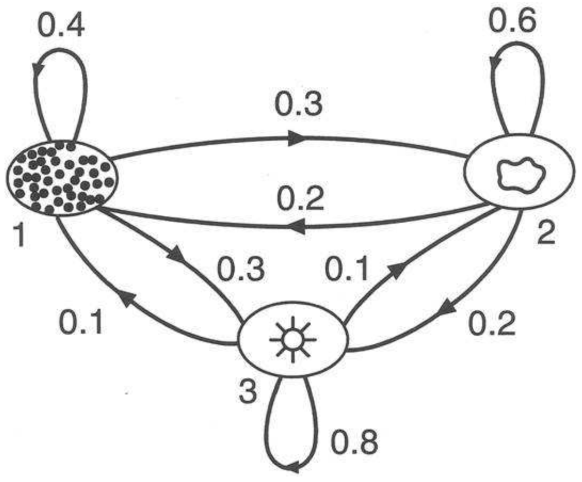


Figure 8.1: State transition of the weather Markov model (from Rabiner 1999).

shows the three probabilities more compactly (notice that their sum is unity). Similarly the probabilities that the cloudy state would transition into the three states can be estimated, and is shown in the second row of the matrix. This 3×3 matrix is called a state transition matrix, and is denoted as $\mathbf{A}$ and the coefficients have the properties $a_{i,j} \geq 0$ and $\sum_j a_{i,j} = 1$ since they obey standard stochastic constraints. Figure 8.1 is called a Markov model.

Formally a Markov model (MM) models a process that goes through a sequence of discrete states, such as notes in a melody. At regular spaced, discrete times, the system undergoes a change of state (possibly back to same state) according to a set of probabilities associated with the state. The time instances for a state change is denoted t and the actual state at time t as $x(t)$. The model is a weighted automaton that consists of:

- A set of N states, $S = \{s_1, s_2, s_3, \ldots, s_N\}$.
- A set of transition probabilities, $\mathbf{A}$, where each $a_{i,j}$ in $\mathbf{A}$ represents the probability of a transition from s_i to s_j. I.e $a_{i,j} = P\left[x(t) = j \,|\, x(t-1) = i\right]$.
- A probability distribution, π, where π_i is the probability the automaton will begin in state s_i, i.e $\pi_i = P(x_1 = i)$, $1 \leq i \leq N$. Notice that the stochastic property for the initial state distribution vector is $\sum_i \pi_i = 1$.
- E, a subset of S containing the legal ending states.

In this model, the probability of transitioning from a given state to another state is assumed to depend only on the current state. This is known as the Markov property.

Given a sequence or a set of sequences of similar kind (e.g., a long list of melodies from a composer) the parameters of the model (the transition probabilities) can readily be estimated. The process of identifying the model parameters is called training the model. In all discussions it is implicitly assumed that the probabilities of transitions are fixed and do not depend on past transitions.

Suppose we are given a Markov model (i.e., $\mathbf{A}$ given). Given an arbitrary state sequence $\mathbf{x} = [x(1), x(2), ..., x(L)]$ we can calculate the probability that $\mathbf{x}$ has been generated by our model. This is given by the product

$$P(\mathbf{x}) = P(x(1)) \times P(x(1) \rightarrow x(2)) \times P(x(2) \rightarrow x(3)) \times \ldots \times P(x(L-1) \rightarrow x(L))$$

where $P(x(1)) = \pi(x(1))$ is the probability that x(1) is the initial state, $P(x(k) \to x(m))$ is the transition probability for going from $x(k)$ to $x(m)$, and can be found from the matrix $\mathbf{A}$. For example with reference to the weather Markov model of equation 8.1, given that the weather on day 1 is sunny (state 3), we can ask the question: What is the probability that the weather for next 7 days will be "sun-sun-rai-rain-sun-cloudy-sun . ."? This probability can be evaluated as

$$\begin{aligned} P &= \pi_3 \cdot a_{33} \cdot a_{33} \cdot a_{31} \cdot a_{11} \cdot a_{13} \cdot a_{32} \cdot a_{13} \\ &= 1(0.8)(0.8)(0.1)(0.4)(0.3)(0.1)(0.2) \\ &= 1.536 \times 10^{-4} \end{aligned}$$

The usefulness of such computation is as follows: given a number of Markov models ($\mathbf{A_1}$ for a composer, $\mathbf{A_2}$ for a second composer, and so forth) and given a melody $\mathbf{x}$, we can calculate the probabilities that this melody is generated by any of these models. The model which gives the highest probability is most likely the model which generated the sequence.

8.1.2 Hidden Markov Models

A hidden Markov model (HMM) is obtained by a slight modification of the Markov model. Thus consider the state diagram shown in Figure 8.1 which shows three states numbered 1, 2, and 3. The probabilities of transitions from the states are also indicated, resulting in the state transition matrix $\mathbf{A}$ shown in equation 8.1. Now we can suppose that we can not observe directly the state, but only a symbol that is associated in a probabilistic way to the state. For example when the weather system is in a particular state, it can output one of four possible symbols L, M, H, VH (corresponding to temperature classes low, medium, high, very high), and there is a probability associated with each of these. This is summarized in the so-called output matrix $\mathbf{B}$

$$\mathbf{B} = \{b_{i,j}\} = \begin{bmatrix} 0.4 & 0.3 & 0.2 & 0.1 \\ 0.2 & 0.5 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.4 & 0.4 \end{bmatrix} \tag{8.2}$$

The element $b_{i,j}$ represents the probability of observing the temperature class j when the weather is in the (non observable) state i, i.e. $b_{i,j} = P(x(t) = s_i | x(t) = j)$. For example when the weather is rainy (state $i = 1$), the probability of measuring medium temperature (output symbol $j = 2$) is $b_{1,2} = 0.3$.

More formally, an HMM requires two things in addition to that required for a standard Markov model:

- A set of possible observations, $O = \{o_1, o_2, o_3, \ldots, o_n\}$.
- A probability distribution $\mathbf{B}$ over the set of observations for each state in S.

8.1.2.0.1 Basic HMM problems In order to apply the hidden Markov model theory successfully there are three problems that need to be solved in practice. These are listed below along with names of standard algorithms which have been developed for these.

1. Learn structure problem. Given an HMM (i.e., given the matrices $\mathbf{A}$ and $\mathbf{B}$) and an output sequence $o(1), o(2), \ldots$, compute the state sequence $x(k)$ which most likely generated it. This is solved by the famous Viterbi's algorithm (see 8.1.4.2).

2. Evaluation or scoring problem. Given the HMM and an output sequence $o(1), o(2), \ldots$ compute the probability that the HMM generates this. We can also view the problem as one of scoring how well a given model matches a given output sequence. If we are trying to choose among several competing models, this ranking allow us to choose the model that best matches the observations. The forward-backward algorithm solves this (see 8.1.4.1).

3. Training problem. How should one design the model parameters $\mathbf{A}$ and $\mathbf{B}$ such that they are optimal for an application, e.g., to represent a melody? The most popular algorithm for this is the expectation maximization algorithm commonly known as the EM algorithm or the Baum-Welch algorithm (see [2] for more details).

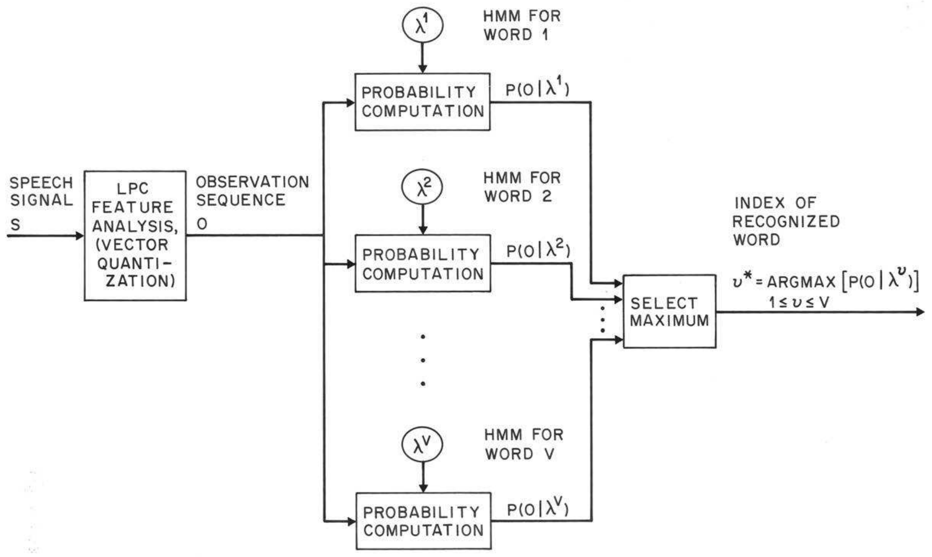


Figure 8.2: Block diagram of an isolated word recognizer (from Rabiner 1999).

For example let us consider a simple isolated word recognizer (see Figure 8.2). For each word we want to design a separate N-state HHM. We represent the speech signal as a time sequence of coded spectral vectors. Hence each observation is the index of the spectral vector closest to the original speech signal. Thus for each word, we have a training sequence consisting of repetitions of codebook indices of the word.

The first task is to build individual word models. This task is done by using the solution to Problem 3 to estimate model parameters for each word model. To develop an understanding of physical meaning of the model state, we use the solution to Problem 1 to segment each of the word state sequence into states, and then study the properties of the spectral vectors that lead to the observations occurring in each state. Finally, once the set of HMMs has been designed, recognition of an unknown word is performed using the solution to Problem 2 to score each word model based on the observation sequence, and select the word whose model score is highest.

We should remark that the HMM is a stochastic approach which models the given problem as a doubly stochastic process in which the observed data are thought to be the result of having passed the true (hidden) process through a second process. Both processes are to be characterized using only the one that could be observed. The problem with this approach, is that one do not know anything about the Markov chains that generate the speech. The number of states in the model is unknown, there probabilistic functions are unknown and one can not tell from which state an observation was produced. These properties are hidden, and thereby the name hidden Markov model.

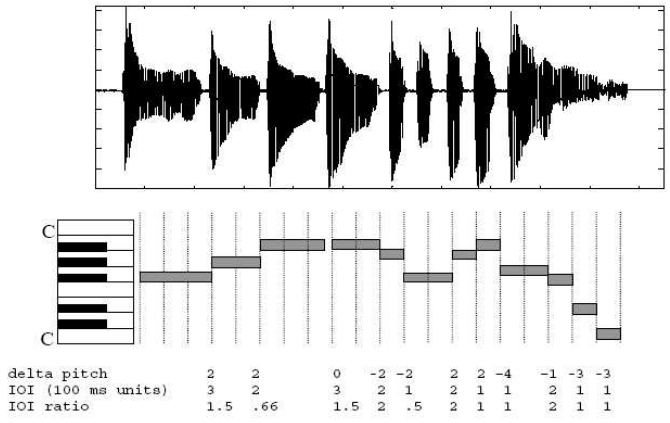
8.1.3 Markov Models Applied to Music

Hiller and Isaacson (1957) were the first to implement Markov chains in a musical application. They developed a computer program that used Markov chains to compose a string quartet comprised of four movements entitled the Illiac Suite. Around the same time period, Meyer and Xenakis (1971) realized that Markov chains could reasonably represent musical events. In his book Formalized Music [4], Xenakis described musical events in terms of three components: frequency, duration, and intensity. These three components were combined in the form of a vector and then were used as the states in Markov chains. In congruence with Xenakis, Jones (1981) suggested the use of vectors to describe notes (e.g., note = pitch, duration, amplitude, instrument) for the purposes of eliciting more complex musical behavior from a Markov chain. In addition, Polansky, Rosenboom, and Burk (1987) proposed the use of hierarchical Markov chains to generate different levels of musical organization (e.g., a high level chain to define the key or tempo, an intermediate level chain to select a phrase of notes, and a low level chain to determine the specific pitches). All of the aforementioned research deals with the compositional aspects and uses of Markov chains. That is, all of this research was focused on creating musical output using Markov chains.

8.1.3.1 HMM models for music search: MuseArt

In the MuseArt system for music search and retrieval, developed at Michigan University by Jonah Shifrin, Bryan Pardo, Colin Meek, William Birmingham, musical themes are represented using a hidden Markov model (HMM).

8.1.3.1.1 Representation of a query. The query is treated as an observation sequence and a theme is judged similar to the query if the associated HMM has a high likelihood of generating the query. A piece of music is deemed a good match if at least one theme from that piece is similar to the query. The pieces are returned to the user in order, ranked by similarity.



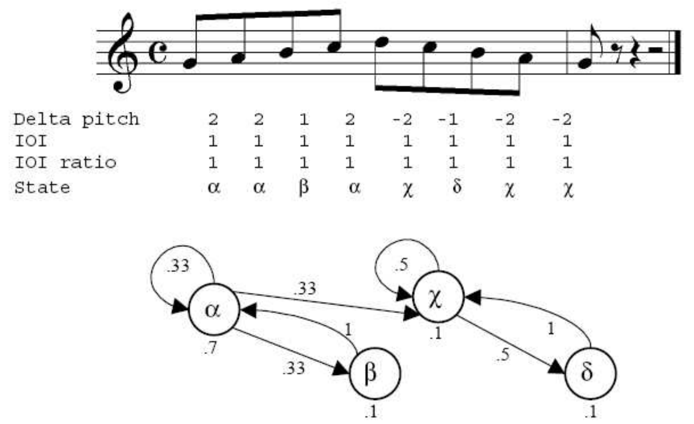
delta pitch	2	2	0	-2	-2	2	2	-4	-1	-3	-3
IOI (100 ms units)	3	2	3	2	1	2	1	1	2	1	1
IOI ratio	1.5	.66	1.5	2	.5	2	1	1	2	1	1

Figure 8.3: A sung query (from Shifrin 2002)

A query is a melodic fragment sung by a single individual. The singer is asked to select one syllable, such as ta or la, and use it consistently during the query. The consistent use of a single consonant-vowel pairing lessens pitch-tracker error by providing a clear onset point for each note, as well as reducing error caused by vocalic variation. A query is recorded as a .wav file and is transcribed into a MIDI based representation using a pitch-tracking system. Figure 8.3 shows a time-amplitude representation of a sung query, along with example pitch-tracker output (shown as piano roll) and a sequence of values derived from the MIDI representation (the $deltaPitch$, IOI and $IOIratio$ values). Time values in the figure are rounded to the nearest 100 milliseconds. We define the following.

- A note transition between note n and note $n + 1$ is described by the duple $< deltaPitch$, $IOIratio >$.
- $deltaPitch_n$ is the musical interval, i.e. the pitch difference in semitones between note n and note $n + 1$.
- $IOIratio_n$ is IOI_n/IOI_{n+1}, where the inter onset interval (IOI_n) is the difference between the onset of notes n and $n + 1$. For the final transition, $IOI_n = IOI_n/duration_{n+1}$.

A query is represented as a sequence of note transitions. Note transitions are useful because they are robust in the face of transposition and tempo changes. The $deltaPitch$ component of a note transition captures pitch-change information. Two versions of a piece played in two different keys have the same $deltaPitch$ values. The $IOIratio$ represents the rhythmic component of a piece. This remains constant even when two performances are played at very different speeds, as long as relative durations within each performance remain the same. In order to reduce The number of possible IOI ratios is reduced by quantizing them to one of 27 values, spaced evenly on a logarithmic scale. A logarithmic scale was selected because data from a pilot study indicated that sung $IOIratio$ values fall naturally into evenly spaced bins in the log domain.



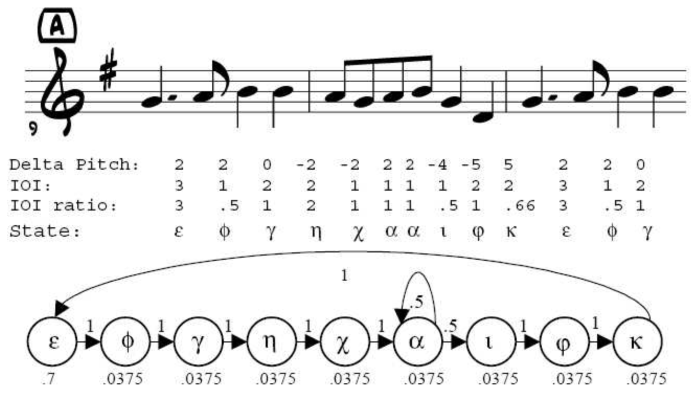
Delta pitch	2	2	1	2	-2	-1	-2	-2
IOI	1	1	1	1	1	1	1	1
IOI ratio	1	1	1	1	1	1	1	1
State	α	α	β	α	χ	δ	χ	χ

Figure 8.4: Markov model for a scalar passage (from Shifrin 2002)

The directed graph in Figure 8.4 represents a Markov model of a scalar passage of music. States are note transitions. Nodes represent states. The numerical value below each state indicates the probability a traversal of the graph will begin in that state. As a default, all states are assumed to be legal ending states. Directed edges represent transitions. Numerical values by edges indicate transition probabilities. Only transitions with non-zero probabilities are shown.

In Markov model, it is implicitly assumed that whenever state s is reached, it is directly observable, with no chance for error. This is often not a realistic assumption. There are multiple possible sources of error in generating a query. The singer may have incorrect recall of the melody he or she is attempting to sing. There may be production errors (e.g., cracked notes, poor pitch control). The transcription system may introduce pitch errors, such as octave displacement, or timing errors due to the quantization of time. Such errors can be handled gracefully if a probability distribution over the set of possible observations (such as note transitions in a query) given a state (the intended note transition of the singer) is maintained. Thus, to take into account these various types of errors, the Markov model should be extended to a hidden Markov Model, or HMM. The HMM allows us a probabilistic map of observed states to states internal to the model (hidden states). In the system, a query is a sequence of observations. Each observation is a note-transition duple, $< deltaPitch, IOIratio >$. Musical themes are represented as hidden Markov models whose states also corresponds to note-transition duples. To make use of the strengths of a hidden Markov model, it is important to model the probability of each observation o_i in the set of possible observations, O, given a hidden state, s.

8.1.3.1.2 Making Markov Models from MIDI. Our system represents musical themes in a database as HMMs. Each HMM is built automatically from a MIDI file encoding the theme. The unique duples characterizing the note transitions found in the MIDI file form the states in the model. FigureFigure 8.4 shows a passage with eight note transitions characterized by four unique duples. Each unique duple is represented as a state. Once the states are determined for the model, transition probabilities between states are computed by calculating what proportion of the time state a follows state b in the theme. Often, this results in a large number of deterministic transitions. Figure 8.5 is an example of this, where only a single state has two possible transitions, one back to itself and the other on to the next state. Note that there is not a one-to-one correspondence between model and observation sequence. A single



Delta Pitch:	2	2	0	-2	-2	2	2	-4	-5	5	2	2	0
IOI:	3	1	2	2	1	1	1	1	2	2	3	1	2
IOI ratio:	3	.5	1	2	1	1	1	.5	1	.66	3	.5	1
State:	ε	ϕ	γ	η	χ	α	α	ι	φ	κ	ε	ϕ	γ

Figure 8.5: Markov model for Alouette fragment (from Shifrin 2002)

model may create a variety of observation sequences, and an observation sequence may be generated by more than one model. Recall that our approach defines an observation as a duple, ¡deltaPitch, IOIratio¿. Given this, the observation sequence $q = \{(2,1),(2,1),(2,1)\}$ may be generated by the HMM in Figure 8.4 or the HMM in Figure 8.5.

8.1.3.1.3 Finding the best target. The themes in the database are coded as HMMs and the query is treated as an observation sequence. Given this, we are interested in finding the HMM most likely to generate the observation sequence. This can be done using the Forward algorithm. The Forward algorithm, given an HMM and an observation sequence, returns a value between 0 and 1, indicating the probability the HMM generated the observation sequence. Given a maximum path length, L, the algorithm takes all paths through the model of up to L steps. The probability each path has of generating the observation sequence is calculated and the sum of these probabilities gives the probability that the model generated the observation sequence. This algorithm takes on the order of $|S|^2L$ steps to compute the probability, where $|S|$ is the number of states in the model.

Let there be an observation sequence (query), O, and a set of models (themes), M. An order may be imposed on M by performing the Forward algorithm on each model m in M and then ordering the set by the value returned, placing higher values before lower. The i-th model in the ordered set is then the i-th most likely to have generated the observation sequence. We take this rank order to be a direct measure of the relative similarity between a theme and a query. Thus, the first theme is the one most similar to the query.

8.1.3.2 Markov sequence generator

Markov models can be thought of as generative models. A generative model describes an underlying structure able to generate the sequence of observed events, called an observation sequence. Note that there is not a one-to-one correspondence between model and observation sequence. A single model may create a variety of observation sequences, and an observation sequence may be generated by more than one model.

A HMM can be used as generator to give an observation sequence O as follow

1. Choose initial state $x(1) = S_1$ according the initial state distribution π.
2. Set $t = 1$
3. Choose $o(t)$ according the symbol probability distribution in state $x(t)$ described in matrix $\mathbf{B}$
4. Transit to new state $x(t+1) = S_j$ according to the state transition probability for state $x(t) = i$, i.e. $a_{i,j}$
5. Set $t = t + 1$ and return to step 2

If a simple Markov model is used as generator, step 3 is skipped, and the state $x(t)$ is used in output.

The ”hymn tunes” of Figure 8.6 were generated by computer from an analysis of the probabilities of notes occurring in various hymns. A set of hymn melodies were encoded (all in C major). Only hymn melodies in 4/4 meter and containing two four-bar phrases were used. The first ”tune” was generated by simply randomly selecting notes from each of the corresponding points in the analyzed melodies. Since the most common note at the end of each phrase was ‘C’ there is a strong likelihood that the randomly selected pitch ending each phrase is C.

8.1.4 Algorithms

8.1.4.1 Forward algorithm

The Forward algorithm is uses to solve the evaluation or scoring problem. Given the HMM $\lambda = (A, B, \Pi)$ and an observation sequence $O = o(1)o(2)\ldots o(L)$ compute the probability $P(O|\lambda)$ that the HMM generates this. We can also view the problem as one of scoring how well a given model matches a given output sequence. If we are trying to choose among several competing models, this ranking allow us to choose the model that best matches the observations. The most straighforward

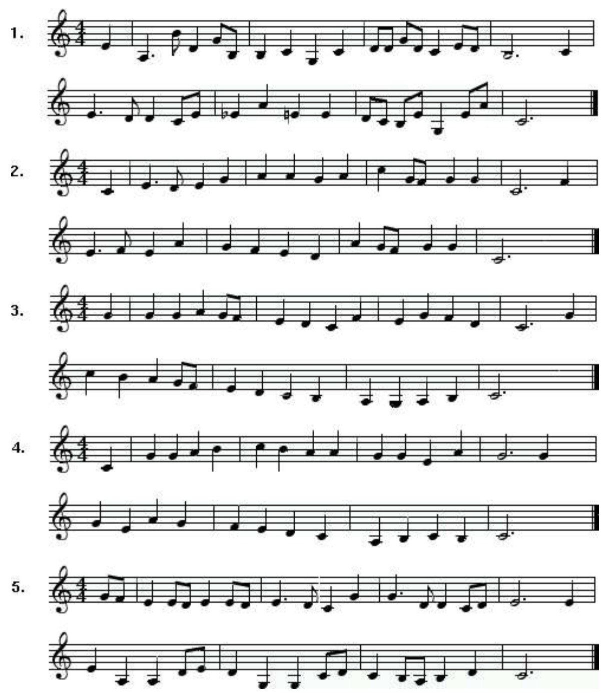


Figure 8.6: "Hymn tunes" generated by computer from an analysis of the probabilities of notes occurring in various hymns. From Brooks, Hopkins, Neumann, Wright. "An experiment in musical composition." IRE Transactions on Electronic Computers, Vol. 6, No. 1 (1957).

procedure is through enumerating every possible state sequence of lenght L (the number of observations), computing the joint probability of the state sequence and O and finally summing the joint probabilty over all possible sate sequence. But if there are N possible states that can be reached, there are N^L possible state sequences and thus such direct approach have exponential computational complexity.

However we can notice that there are only N states and we can apply a dynamic programming stategy. To this purpose let us define the forward variable $\alpha_t(i)$ as

$$\alpha_t(i) = P(o(1)o(2)\dots o(t),\, x(t) = s_i|\lambda)$$

i.e. the probability of the partial observation $o(1)o(2)\dots o(t)$ and state s_i at time t, given the model λ. The Forward algorithm solves the problem with a dynamic programming strategy, using an iteration on the sequence length (time t), as follows:

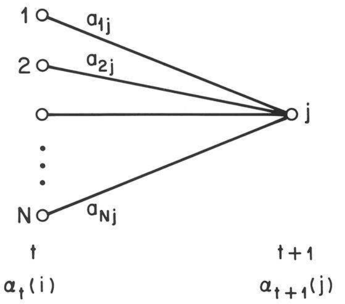
1. Initialization
$$\alpha_1(i) = \pi(i) b_i(o_1),\ 1 \leq i \leq N$$

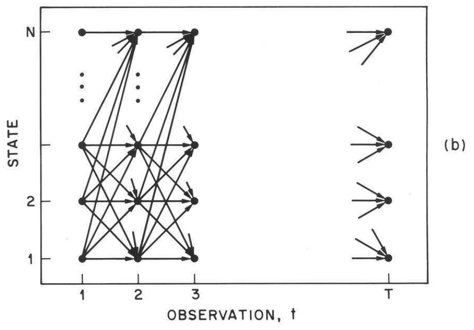
2. Induction
$$\alpha_{t+1}(i) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1}) \quad 1 \leq t \leq L-1$$
$$1 \leq i \leq N$$

3. Termination

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_L(i)$$

Step 1) initializes the forward probabilities as the joint probability of state i and initial observation $o(1)$. The induction step is illustrated in Figure 8.7(a). This figure shoes that state s_j can be reached at time $t+1$ from the N possible states at time t. Since $\alpha_t(i)$ is the probability that $o(1)o(2)\dots o(t)$ is observed and $x(t) = s_i$, the product $\alpha_t(i)a_{ij}$ is the probability that $o(1)o(2)\dots o(t)$ is observed and state s_j is reached at time $t+1$ via state s_i at time t. Summing this product over all the possible states results in the probability of s_j with all the previous observations. Finally $\alpha_{t+1}(i)$ s obtained by accounting for observation o_{t+1} in state s_j, i.e. by multiplying by the probability $b_j(o_{t+1})$. Finally step 3) gives the desired $P(O|\lambda)$ as the sum of the terminal forward variables $\alpha_L(i)$. In fact $\alpha_L(i)$ is the probability of the observed sequence and that the system at time $t = L$ is in the state s_i. Hence $P(O|\lambda)$ is just the sum of the $\alpha_L(i)$'s. The time computational complexity of this algorithm is $O(N^2L)$. The forward probability calculation is based upon the lattice structure shown in figure 8.7(b). The key is that since there are only N states, all the possible state sequences will remerge into these N nodes, no matter how long the observation sequence. Notice that the calculation of $\alpha_t(i)$ involves multiplication with probabilities. All these probabilities have a value less than 1 (generally significantly less than 1), and as t starts to grow large, each term of $\alpha_t(i)$ starts to head exponentially to zero, exceed the precision range of the machine. To avoid this problem, a version of the Forward algorithm with scaling should be used. See [2] for more details.





(a) (b)

Figure 8.7: (a) Illustration of the sequence of operations required for the computation of the forward variable $\alpha_{t+1}(i)$. (b) Implementation of the computation of $\alpha_{t+1}(i)$ in terms of a lattice of observation t and states i.

8.1.4.2 Viterbi algorithm

The Viterbi algorithm, based on dynamic programming, is used to solve the structure learning problem. Given an HMM λ (i.e., given the matrices $\mathbf{A}$ and $\mathbf{B}$) and an output sequence $O = \{o(1)o(2)\dots o(L)\}$, find the *single* best state sequence $X = \{x(1)x(2)\dots x(L\}$ which most likely generated it. To this purpose we define the quantity

$$\delta_t(i) = P\left[x(1)x(2)\dots x(t) = s_i, o(1)o(2)\dots o(t)\,|\lambda\right]$$

i.e. $\delta_t(i)$ is the best score (highest probability) along a single path at time t, which accounts for the first t observations and ends in state s_i. By induction we have

$$\delta_{t+1}(i) = \max_i \left[\delta_t(i)a_{ij}\right] b_j(o_{t+1})$$

To actually retrieve the state sequence, we need to keep track of the argument which maximized the previous expression, for each t and j using a predecessor array $\psi_t(j)$. The complete procedure of Viterbi algorithm is

```
1. Initialization
   for $1 \le i \le N$
           $\delta_1(i) = \pi(i)b_i(o_1)$
           $\psi_1(i) = 0$

2. Induction
   for $1 \le t \le L-1$
           for $1 \le j \le N$
                   $\delta_{t+1}(j) = \max_i \left[\delta_t(i)a_{ij}\right] b_j(o_{t+1})$
                   $\psi_{t+1}(j) = \text{argmax}_i \left[\delta_t(i)a_{ij}\right]$

3. Termination
   $P^* = \max_i \left[\delta_T(i)\right]$
   $x^*(T) = \text{argmax}_i \left[\delta_T(i)\right]$

4. Path backtracking
   for $t = L-1$ downto 1
           $x^*(t) = \psi_{t+1}(x^*_{t+1})$
```

Notice that the structure of Viterbi algorithm is similar in implementation to forward algorithm. The major difference is the maximization over the previous states which is used in place of the summing procedure in forward algorithm. Both algorithms used the lattice computational structure of figure 8.7(b) and have computational complexity N^2L. Also Viterbi algorithm presents the problem of multiplicationof probabilities. One way to avoid this is to take the logarithm of the model parameters, giving that the multiplications become additions. The induction thus becomes

$$\log[\delta_{t+1}(i)] = \max_i(\log\left[\delta_t(i)\right] + \log\left[a_{ij}\right] + \log\left[b_j(o_{t+1})\right]$$

Obviously will this logarithm become a problem when some model parameters has zeros present. This is often the case for A and π and can be avoided by adding a small number to the matrixes. See [2] for more details.

To get a better insight of how the Viterbi (and the alternative Viterbi) works, consider a model with $N = 3$ states and an observation of length $L = 8$. In the initialization ($t = 1$) is $\delta_1(1)$, $\delta_1(2)$ and $\delta_1(3)$ found. Lets assume that $\delta_1(2)$ is the maximum. Next time ($t = 2$) three variables will be used namely $\delta_2(1)$, $\delta_2(2)$ and $\delta_2(3)$. Lets assume that $\delta_2(1)$ is now the maximum. In the same manner will the following variables $\delta_3(3)$, $\delta_4(2)$, $\delta_5(2)$, $\delta_6(1)$, $\delta_7(3)$ and $\delta_8(3)$ be the maximum at their time, see Fig.8.8. This algorithm is an example of what is called the Breadth First Search (Viterbi employs this essentially). In fact it follows the principle: ”Do not go to the next time instant $t + 1$ until the nodes at at time T are all expanded”.

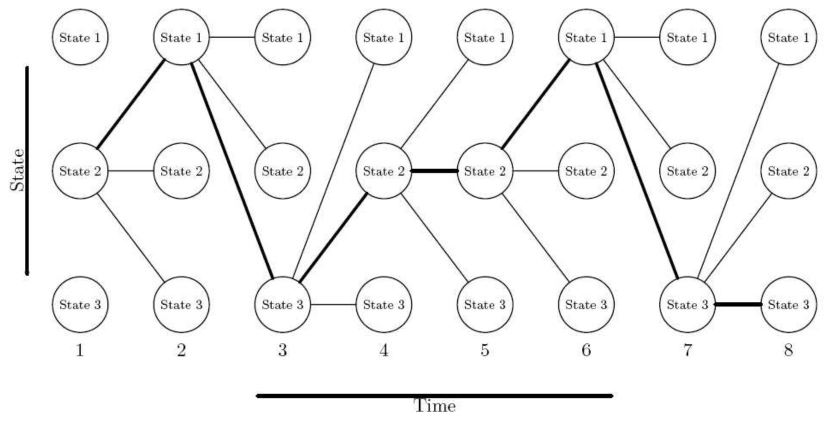


Figure 8.8: Example of Viterbi search.

8.2 Algorithmic Composition

Composers have long been fascinated by mathematical concepts in relation to music. The concept of "music of the spheres," dating back to Pythagoras, held the notion that humans were governed by the perfect proportions of the natural universe. This mathematical order may be seen in the musical interval choice and system of organization that was used by the ancient Greek culture.

Procedures that entail rules or provisions to govern the act of musical composition have been used since the Medieval period; these same principles have been applied in very specific methods to many of the recent computer programs developed for algorithmic composition.

Algorithmic composition could be described as "a sequence (set) of rules (instructions, operations) for solving (accomplishing) a [particular] problem (task) [in a finite number of steps] of combining musical parts (things, elements) into a whole (composition)". From this definition we can see that it is not necessary to use computers for algorithmic composition as we often infer; Mozart did not when he described the Musical Dice Game.

The concept of algorithmic composition is not something new. Pythagoras (around 500 B.C.) believed that music and mathematics were not separate studies. Hiller and Isaacson (1959) were probably the first who used a computational model using random number generators and Markov chains for algorithmic composition. Since then many researchers have tried to address the problem of algorithmic composition from different points of view.

Some of the algorithmic programs and compositions specify score information only. Score information includes pitch, duration, and dynamic material, whether written for acoustic and/or electronic instruments. That is, there are instances in which a composer makes use of a computer program to generate the score while the instrumental selection has been predetermined as either an electronic orchestra or a realization for acoustic instruments. Other algorithmic programs specify both score and electronic sound synthesis. In this instance, the program is used not only to generate the score, but also the electronic timbres to be used in performance.

8.2.1 Categories of algorithmic processes

A review can not be exhaustive because there have been so many attempts. In the following subsections [1] we give some representative examples of systems which employ different methods which we categorise, based on their most prominent feature, as follows:

Mathematical models Stochastic processes and especially Markov chains have been used extensively in the past for algorithmic composition (e.g., Xenakis, 1971).

One manner of statistical analysis that has been frequently used in musical composition is Markov Analysis or Markov Chains. Named for the mathematician Andre Andreevich Markov (1856-1922), Markov Chains were conceived as a means by which decisions could be made based on probability. Information is linked together in a series of states based on the probability that state A will be followed by state B. The process is continually in transition because state A is then replaced by state B which continues to look at the probability of being followed by yet another state B. The so-called orders of Markov Analysis indicate the relationship between states. For instance, zeroth-order analysis assumes that there are no relationships between states; that is, the relationship between any two states is random. First-order analysis simply counts the frequency with which specific states occur within the given data. Second-order analysis examines the relationships between any two consecutive states (e.g., what is the probability that the state B would follow state A?). Third-order analysis determines the probability of three consecutive states occurring in a row (e.g., what is the probability that state A would be followed by state B, would be followed by state C?). Fourth-order analyzes the chance of four states following each other. Composer/scientist Lejaren Hiller made use of Markov Chains, statistical analysis, and stochastic procedures in algorithmic composition beginning in the late 1950s.

Probably the most important reason for stochastic precesses is their low complexity which makes them good candidates for real-time applications.

We also see computational models based on chaotic nonlinear systems or iterated functions but it is difficult to judge the quality of their output, because, unlike all the other approaches, their ”knowledge” about music is not derived from humans or human works. Since the 1970s basic principles of the irregularities in nature have been studied by the scientific community, and by the 1980s chaos was the focus of a great deal of attention. The new science has spawned its own language, an elegant shop talk of fractals and bifurcations, intermittencies and periodicities, folded-towel diffeomorphisms and smooth noodle maps. . . To some physicists chaos is a science of process rather than state, or becoming rather than being. One subcategory of chaotic structures that has come to the forefront since ca. 1975 is fractals. Fractals are recursive and produce ’parent-child’ relationships in which the offspring replicate the initial structure. Seen in visual art as smaller and smaller offshoots from the original stem, fractals were categorized by Benoit Mandelbrot in his book, The Fractal Geometry of Nature. The underlying principles of chaos may best be thought of in terms of natural, seemingly disorderly designs.

> ”Nature forms patterns. Some are orderly in space but disorderly in time, others orderly in time but disorderly in space. Some patterns are fractal, exhibiting structures self-similar in scale. Others give rise to steady states or oscillating ones. Pattern formation has become a branch of physics and of materials science, allowing scientists to model the aggregation of particles into clusters, the fractured spread of electrical discharges, and the growth of crystals in ice and metal alloys.”

[1]adapted from Papadopoulos, Wiggins 1993

The main disadvantages of stochastic processes are:

- Someone needs to find the probabilities by analysing many pieces. Something necessary if we want to simulate one style. The resulting models will only generate music of similar style to the input.
- For higher order Markov models, transition tables become unmanageably large for the average computer. While many techniques exist for a more compact representation of sparse matrices (which usually result for higher order models), these require extra computational effort that can hinder real time performance.
- The deviations from the norm and how they are incorporated in the music is an important aspect. They also provide little support for structure at higher levels (unless multiple layered models are used where each event represents an entire Markov model in itself).

Knowledge based systems In one sense, most AI systems are knowledge based systems (KBS). Here, we mean systems which are symbolic and use rules or constraints. The use of KBS in music seems to be a natural choice especially when we try to model well defined domains or we want to introduce explicit structures or rules. Their main advantage is that they have explicit reasoning; they can explain their choice of actions.

Even though KBS seem to be the most suitable choice, as a stand alone method, for algorithmic composition they still exhibit some important problems:

- Knowledge elicitation is difficult and time consuming, especially in subjective domains such as music.
- Since they do what we program them to do they depend on the ability of the ”expert”,who in many cases is not the same as the programmer, to clarify concepts, or even find a fiexible representation.
- They become too complicated if we try to add all the exceptions to the rule and their preconditions, something necessary in this domain.

Grammars The idea that there is a grammar of music is probably as old as the idea of grammar itself.

Linguistics is an attempt to identify how language functions: what are the components, how do the components function as a single unit, and how do the components function as single entities within the context of the larger unit. Linguistic theory models this unconscious knowledge [of speech] by a formal system of principles or rules called a generative grammar, which describes (or ’generates’) the possible sentences of the language. Curtis Roads has made a distinction between the specific use of generative grammars and the more open-ended field of algorithmic composition in that ”Generative modeling of music can be distinguished from algorithmic composition on the basis of different goals. While algorithmic composition aims at an aesthetically satisfying new composition, generative modeling of music is a means of proposing and verifying a theory of an extant corpus of compositions or the competence that generated them.”

Experiments in Musical Intelligence (EMI) is a project focused on the understanding of musical style and stylistic replication of various composers (Cope, 1991, 1992). EMI needs as an input a minimum of two works from which extracts ”signatures” using pattern matching. The meaningful arrangement of these signatures in replicated works is accomplished through the use of an augmented transition network (ATN).

Some basic problems of the grammars are:

- They are hierarchical structures while much music is not (i.e improvisation). Therefore ambiguity might be necessary since it ”can add to the representational power of a grammar”.
- Most, if not all, musical grammar implementations do not make any strong claims about the semantics of the pieces.
- Usually a grammar can generate a large number musical strings of questionable quality.
- Parsing is, in many cases, computationally expensive especially if we try to cope with ambiguity.

Evolutionary methods Genetic algorithms (GAs) have proven to be very efficient search methods, especially when dealing with problems with very large search spaces. This, coupled with their ability to provide multiple solutions, which is often what is needed in creative domains, makes them a good candidate for a search engine in a musical application. We can divide the following attempts into two categories based on the implementation of the fitness function.

Use of an objective fitness function. In this case the chromosomes are evaluated based on formally stated, computable functions. The efficacy of the GA approach depends heavily on the amount of knowledge the system possesses; even so GAs are not ideal for the simulation of human musical thought because their operation in no way simulates human behaviour.

Use of a human as a fitness function. Usually we refer to this type of GA as interactive-GA (IGA). In this case a human replaces the fitness function in order to evaluate the chromosomes.

These attempts exhibit two main drawbacks associated with all IGAs:

- Subjectivity
- Efficiency, the fitness bottleneck. The user must hear all the potential solutions in order to evaluate a population.

Moreover, this approach tells us little about the mental processes involved in music composition since all the reasoning is encoded inaccessibly in the users mind. Most of these approaches exhibit very simple representations in an attempt to decrease the search space, which in some cases compromises their output quality.

Systems which learn In the category of learning systems are systems which, in general, do not have a priori knowledge (e.g. production rules, constraints) of the domain, but instead learn its features by examples. We can further classify these systems, based on the way they store the information, to subsymbolic/distributive (Artificial Neural Networks, ANN) and symbolic (Machine Learning, ML).

ANNs offer an alternative for algorithmic composition to the traditional symbolic AI methods, one which loosely resembles the activities in the human brain, but at the moment they do not seem to be as efficient or as practical, at least as a stand-alone approach. Some of their disadvantages are:

- Composition as compared with cognition is a much more highly intellectual process (more ”symbolic”).The output from a ANN matches the probability distribution of the sequence set to which it is exposed, something which is desirable, but on the other hand shows us its limit: ”While ANNs are capable of successfully capturing the surface structure of a melodic passage and produce new melodies on the basis of the thus acquired knowledge, they mostly fail to pick up the higher-level features of music, such as those related to phrasing or tonal functions”.

- The representation of time can not be dealt efficiently even with ANNs which have feedback. Usually they solve toy problems, with many simplifications, when compared with the knowledge based approaches.
- They can not even reproduce fully the training set and when they do this it might mean that they did not generalise.
- Even though it seems exciting that a system learns by examples this is not always the whole truth since the human in many cases needs to do the ”filtering” in order not to have in the training set examples which conflict.
- Usually, the researchers using ANNs say that their advantage over knowledge based approaches is that they can learn from examples things which can’t be represented symbolically using rules (i.e. the ”exceptions”).

Hybrid systems Hybrid systems are ones which use a combination of AI techniques. In this section we discuss systems which combine evolutionary and connectionist methods, or symbolic and subsymbolic ones. The reason behind using hybrid systems, not only for musical applications, is very simple and logical. Since each AI method exhibits different strengths then we should adopt a postmodern attitude by combining them.

The main disadvantage of hybrid systems is that they are usually complicated, especially in the case of tightly-coupled or fully integrated models. The implementation, verification and validation is also time consuming.

8.2.2 Discussion

First there is usually no evaluation of the output by real experts (e.g., professional musicians) in most of the systems and second, the evaluation of the system (algorithm) is given relatively small consideration

8.2.2.0.1 Knowledge representation Two almost ubiquitous issues in AI are representation of knowledge and search method. From one point of view, our categorisation above, reflects the search method, which however, constrains the possible representations of knowledge. For example structures which are easily represented symbolically are often difficult to represent with a ANN.

In many AI systems, especially symbolic, the choice of the knowledge representation is an important factor in reducing the search space. For example Biles (1994) and Papadopoulos and Wiggins (1998) used a more abstract representation, representing the degrees of the scale rather than the absolute pitches. This reduced immensely the search space since the representation did not allow the generation of non-scale notes (they are considered dissonant) and the inter-key equivalence was abstracted out.

Most of the systems reviewed exhibit a single, fixed representation of the musical structures. Some, on the other hand, use multiple viewpoints which we believe simulate more closely human musical thinking.

8.2.2.0.2 Computational Creativity Probably the most difficult task is to incorporate in our systems the concept of creativity. This is difficult since we do not have a clear idea of what creativity is (Boden, 1996).

Some characteristics of computational creativity, which were proposed by Rowe and Partridge (1993) are:

- Knowledge representation is organised in such a way that the number of possible associations is maximised. A flexible knowledge representation scheme. Similarly Boden (1996) says that representation should allow to explore and transform the conceptual space.
- Tolerate ambiguity in representations.
- Allow multiple representations in order to avoid the problem of ”functional fixity”.
- The usefulness of new combinations should be assessable.

New combinations need to be elaboratable to find out their consequences. One question that AI researchers should aim to answer is: do we want to simulate human creativity itself or the results of it? (Is DEEP BLUE creative, even if it does not simulate the human mind?) This is more or less similar to the, subtle in many cases, distinction between cognitive modeling and knowledge engineering.

8.3 Gestural Control of Sound Synthesis

adapted from Wandeley 2004

8.3.1 Introduction

The evolution of computer music has brought to light a plethora of sound synthesis methods available in general and inexpensive computer platforms, allowing a large community direct access to real-time computer-generated sound. Both signal and physical models have reached a point where they can be used in concert situations, although much research continues to be carried on in the subject, constantly bringing innovative solutions and developments.

On the other hand, input device technology that captures different human movements can also be viewed as in an advanced stage, considering both noncontact movements and manipulation. Specifically regarding manipulation, tactile and force feedback devices for both nonmusical2 and musical contexts have already been proposed. We are then in a stage where such devices and sound synthesis methods can be combined to create new computer-based musical instruments, or digital musical instruments (DMI), producing gesturally controlled real time computer-generated sound. The ultimate goal is to design new DMIs capable of obtaining similar levels of control subtlety as those available in acoustic instruments, but at the same time extrapolating the capabilities of existing instruments.

In short, we need to devise ways to interact with computers in a musical context, i.e., to control multiple continuous parameters that allow the generation of sound in real time. This topic amounts to a branch of knowledge known as HCI. Various questions need to be addressed, such as the following

- Which are the specific constraints that exist in the musical context with respect to general HCI?
- Given the various contexts related to interaction in sound generation systems, what are the similarities and differences within these contexts (interactive installations, DMI manipulation, dancemusic interfaces)?

- How to design systems for these various musical contexts? Which system characteristics are common and which are context specific?

8.3.1.1 HCI and Music

More specifically, gestural control of computer-generated sound can be seen as a highly specialized branch of HCI involving the simultaneous control of multiple parameters, timing, rhythm, and user training.Hunt and Kirk consider various attributes that are characteristic of real-time multi-parametric control systems.

- There is no fixed ordering to the humancomputer dialogue.
- There is no single permitted set of options (e.g., choices from a menu) but rather a series of continuous controls.
- There is an instant response to the users movements.
- The control mechanism is a physical and multipara-metric device which must be learned by the user until the actions become automatic.
- Further practice develops increased control intimacy and, thus, competence of operation.
- The human operator, once familiar with the system, is free to perform other cognitive activities while operating the system (like talking while driving a car).

8.3.1.2 Interaction in a Musical Context

In order to take into account the specifics of musical interaction, one needs to consider the various existing contexts (sometimes called metaphors for musical control) where gestural control can be applied to computer music.

These different interaction contexts are the result of the evolution of electronic technology allowing, for instance, a same input device to be used in different situations: to generate sounds (notes) or to control the temporal evolution of a set of prerecorded notes. These two contexts traditionally correspond to two separate roles in music, those of the performer and the conductor, respectively. Technology has blurred the difference between traditional roles and allowed novel metaphors derived from other areas, such as HCI.

In this section, we will focus on instrument manipulation, or performer-instrument interaction in the context of real-time sound synthesis control. The approach consists on dividing the subject of gestural control of sound synthesis in four parts]:

- definition and typologies of gesture;
- gesture acquisition and input device design;
- synthesis algorithms;
- mapping of gestural variables to synthesis variables.

The goal is to analyze all four parts, which are equally important to the design of new DMIs.

8.3.2 Control of digital musical instrument

The term digital musical instrument is used to represent an instrument that includes a separate gestural interface (or gestural controller unit) from a sound generation unit. Both units are independent and related by mapping strategies.This is shown in Fig. 8.9.

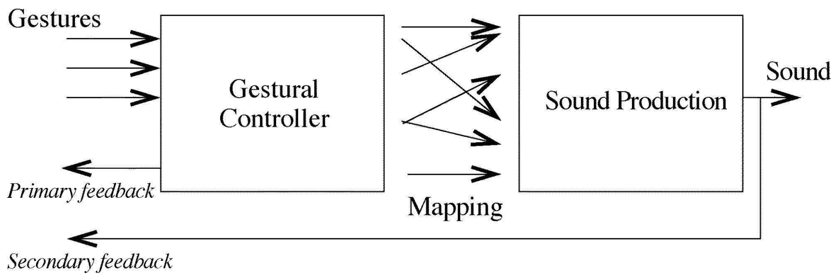


Figure 8.9: A symbolic representation of a DMI.

The term gestural controller4 can be defined here as the input part of the DMI, where physical interaction with the player takes place. Conversely, the sound generation unit can be seen as the synthesis algorithm and its input parameters. The mapping layer refers to the liaison strategies between the outputs of the gestural controller and the input controls of the synthesis algorithm.

This separation is most of the time impossible in the case of acoustic instruments, where the gestural interface is also part of the sound generation unit. If one considers, for instance, a clarinet, the reed, keys, holes, etc., are at the same time both the gestural interface (where the performer interacts with the instrument) and the elements responsible for sound generation. The idea of a DMI is analogous to splitting the clarinet in a way where one could separate these two functions (gestural interface and sound generator) and use them independently.

Clearly, this separation of the DMI into two independent units is potentially capable of extrapolating the functionalities of a conventional musical instrument, the latter tied to physical constraints. On the other hand, basic interaction characteristics of existing instruments may be lost and/or difficult to reproduce, such as tactile/force feedback.

In order to devise strategies concerning the design of new DMIs for gestural control of sound synthesis, it is essential to analyze the characteristics of actions produced by expert instrumentalists during performance. These actions are commonly referred to as gestures in the musical domain. In order to avoid discussing all nuances of the meaning of gesture, let us initially consider performer gestures as performer actions produced by the instrumentalist during a performance, meaning both actions such as prehension and manipulation, and noncontact movements. A detailed discussion is presented.

The importance of the study of gestures in DMI design can be justified by the need to better understand physical actions and reactions that take place during expert performance. Furthermore, gesture information can also be considered as a form of signal, i.e., they can be processed, transformed, and stored using gesture editors.Gestures can also be synthesized using various models of movement or using rules in a similar way to speech synthesis.

In fact, instrumentalists simultaneously execute various types of gestures during performance. Some of them are necessary for the generation of sound, while others are not although the later are also present in most highly skilled instrumentalists performances.

One can approach the study of gestures in a musical context by either analyzing the possible functions of a gesture during performance or by analyzing the physical properties of the gestures taking place. By identifying gestural characteristics functional, in a specific context, or physiological one can ultimately gain insight into the design of gestural acquisition systems.

Regarding both approaches, one fundamental aspect is the existing feedback available to the performer, be it visual, auditory, or tactile-kinesthetic. Feedback can also be considered, depending on its characteristics, as follows.

- Primary/secondary, where primary feedback encompasses visual, auditory (clarinet key noise, for instance), and tactile-kinesthetic feedback, and secondary feedback relates to the sound produced by the instrument.
- Passive/active, where passive feedback relates to feedback provided through physical characteristics of the system (a switch noise, for instance) and active feedback is the one produced by the system in response to a certain user action (sound produced by the instrument).

8.3.2.1 Gestural Acquisition

Once the characteristics of gestures are known, it is essential to devise an acquisition system that will capture these characteristics. In the case of performerinstrument interaction, this acquisition may be performed in three ways.

Direct acquisition, where one or various sensors are used to monitor performers actions. The signals from these sensors present isolated basic physical features of a gesture: pressure, linear or angular displacement, speed, or acceleration. Each physical variable of the gesture to be captured will normally require a different sensor.

Indirect acquisition, where gestures are extracted from the structural properties of the sound produced by the instrument [33][38].Signal processing techniques can then be used in order to derive performers actions by the analysis of the fundamental frequency of the sound, its spectral envelope, its temporal envelope, etc.

Physiological signal acquisition, the analysis of physiological signals, such as EMG. Commercial systems have been developed based on the analysis of muscle tension and used in musical contexts.

8.3.2.1.1 Direct Acquisition. Direct acquisition is performed by the use of different sensors to capture performer actions. Depending on the type of sensors and on the combination of different technologies in various systems, different movements may be tracked.

According to B. Bongers: Sensors are the sense organs of a machine. Sensors convert physical energy (from the outside world) into electricity (into the machine world). There are sensors available for all known physical quantities, including the ones humans use and often with a greater range. For instance, ultrasound frequencies (typically 40 kHz used for motion tracking) or light waves in the infrared frequency range. Tactile-kinesthetic, or tactual,feedback is composed of the tactile and proprioceptive senses.

Direct acquisition has the advantage of simplicity when compared to indirect acquisition, i.e., one can obtain independent streams of data representing individual control parameters. On the other hand, due to the independence of the variables captured, direct acquisition techniques may underestimate the interdependency of the various variables obtained.

8.3.2.1.2 Sensor Characteristics and Musical Applications. Some authors consider that most important sensor characteristics are sensitivity, stability, and repeatability. Other important characteristic relates to the linearity and selectivity of the sensors output, its sensitivity to ambient conditions, etc. A more complete analysis proposes six descriptive parameters applicable to sensors: accuracy, error, precision, resolution, span, and range.

In general instrumentation circuits, sensors typically need to be both precise and accurate, and present a reasonable resolution. In the musical domain, it is often stressed that the choice of a transducer technology matching a specific musical characteristic relates to human performance and perception: for instance, mapping of the output of a sensor that is precise but not accurate to a variable controlling loudness may be satisfactory, but if it is used to control pitch, its inaccuracy will probably be more noticeable.

In music, the use of commercially available sensors developed for other uses is the rule. Only a few researchers have proposed sensors specifically designed for musical use, for instance. Various texts describe different sensors and transducer technologies for general and musical applications.

8.3.2.1.3 Analog-to-Digital Conversion: For the case of gesture acquisition with the use of various sensors, the signals obtained at the sensors outputs are usually available in an analog format, basically in the form of voltage or current signals. In order to be able to use these signals as computer inputs, they need to be sampled and converted in a suitable format, usually Musical Instrument Digital Interface (MIDI) or more advanced protocols such as Open Sound Control (OSC). Various analog-to-MIDI converters have been proposed and are widely available commercially. The first examples had already been developed in the 1980s.

Concerning the various discussions on the advantages and drawbacks of the MIDI protocol and its use, strictly speaking, nothing forces someone to use MIDI or prevents the use of faster or different protocols. As already pointed out, the limiting factor regarding speed and resolution is basically the specifications of the MIDI protocol, not the electronics involved in the design.

It is interesting to notice that many existing systems have used communication protocols other than MIDI in order to avoid speed and resolution limitations. One such system is the *transducteur gestuel rtroactif (TGR)* from ACROE. Other papers have proposed different options to implement gesture acquisition interfaces, such as using hardware initially designed for audio processing.

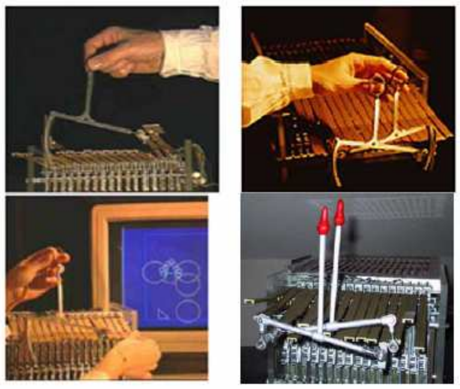


Figure 8.10: *Transducteur gestuel rtroactif (TGR)* from ACROE.

8.3.2.1.4 Indirect Acquisition: As opposed to direct acquisition, indirect acquisition provides information about performer actions from the evolution of structural properties of the sound being produced by an instrument. In this case, the only sensor is a microphone, i.e., a sensor measuring pressure or gradient of pressure. Due to the complexity of the information available in the instruments sound captured by a microphone, various real-time signal processing techniques are used in order to distinguish the effect of a performers action from other factors, e.g., the influence of the acoustical properties of the room or the intrinsic properties of the instrument.

Generically, one could identify basic sound parameters to be extracted in real-time as follows.

- Short-time energy, related to the dynamic profile of the signal, indicates the dynamic level of the sound but also possible differences of the instrument position with respect to the microphone.
- Fundamental frequency, related to the sounds melodic profile, gives information about fingering, for instance.
- Spectral envelope, representing the distribution of sound partial amplitudes, may give information about the resonating body of the instrument.
- Amplitudes, frequencies, and phases of sound partials that can alone provide much of the information obtained by the previous parameters.

8.3.2.1.5 Sampling Gestural Signals: Obviously, in order to perform the analysis of the above or other parameters during direct or indirect acquisition, it is important to consider the correct sampling of the signal. According to the Nyquist theorem, this frequency needs to be at least twice as high as the maximum frequency of the signal to be sampled.

Although one could reasonably consider that frequencies of performer actions can be limited to a few hertz, fast actions can potentially present higher frequencies. A typical sampling frequency for gestural acquisition is 200 Hz. Some systems may use higher values, up to 1 kHz and other researchers considered the ideal sampling frequency to be around 4 kHz.

8.3.2.2 Gestural Controllers

Once one or several sensors are assembled as part of a unique device, this device is called a gestural controller.

As cited above, the gestural controller is the part of the DMI where physical interaction takes place. Physical interaction here means the actions of the performer, be they body movements, empty-handed gestures, or object manipulation, and the perception by the performer of the instruments status and response by means of tactile-kinesthetic, visual, and auditory senses.

Due to the large range of human actions to be captured by the controller7 and depending on the interaction context where it will be used, its design may vary from case to case. Existing controller designs can be classified as follows:

- Instrument-like controllers (see Fig. 2), where the input device design tends to reproduce each feature of an existing (acoustic) instrument in detail. Many examples can be cited, such as electronic keyboards, guitars, saxophones, marimbas, and so on.
- Instrument-inspired controllers that although largely inspired by the existing instruments design, are conceived for another use. Fig. 3 presents one example of such controller, the Super-Polm violin developed by S. Goto, A. Terrier, and P. Pierrot [63], [64], where the input device is loosely based on a violin shape, but is used as a general device to control granular synthesis.

- Extended instruments are instruments augmented by the addition of extra sensors.Commercial augmented instruments included the Yamaha Disklavier, used, for instance, in pieces by J.-C. Risset. Other examples include the flute and the trumpet ,but any existing acoustic instrument may be extended to different degrees by the addition of sensors.

- Alternate controllers (see, e.g., Fig. 4), whose design does not follow that of an established instrument. Some examples include the Hands, graphic drawing tablets (cf. Fig. 5), etc. For instance, an unorthodox gestural controller using the shape of the oral cavity has been proposed.

For instrument-like controllers, although mostly representing a simplified (first-order) model of the acoustic instrument, many of the gestural skills developed by the performer on the acoustic counterparts can be readily applied to the controller. Conversely, for a nonexpert performer, these controllers present roughly the same constraints as those of an acoustic instrument, i.e., technical difficulties inherent to the former will have to be overcome by the nonexpert performer.

Alternate controllers, on the other hand, allow the use of other gesture vocabularies than those of traditional instrument manipulation, thus being in principle less demanding for nonexpert performers. Even so, performers still have to develop specific skills for mastering these new gestural vocabularies.

These controllers can furthermore be classified into different categories. This fact can be modified by the use of different mapping strategies.

- Touch, expanded range,or immersive controllers [76],depending on the amount of physical contact required from the performer. Mulder also separates immersive controllers into internal, external, and symbolic controllers according to the possibilities of visualization of the control surface. In a different approach, Piringer classifies immmersive controllers into partial or completely immersive controllers.

- Individual or collaborative controllers, depending on whether the instrument is performed by one or multiple performers at one time.

- Metaphorical or ad hoc controllers, and so on.

8.3.3 Mapping of gestural variables to synthesis parameters

Once gesture variables are available either from independent sensors or as a result of signal analysis techniques in the case of indirect acquisition, one then needs to relate these output variables to the available synthesis input variables.

Depending on the sound synthesis method to be used, the number and characteristics of these input variables may vary. For signal model methods, one may have: 1) amplitudes, frequencies, and phases of sinusoidal sound partials for additive synthesis; 2) an excitation frequency plus each formants center frequency, bandwidth, amplitude, and skew for formant synthesis; 3) carrier and modulation coefficients ($f_c : f_m$ ratio) for FM synthesis, etc. It is clear that the relationship between the gestural variables and the synthesis inputs available is far from obvious.

For the case of physical models, the available input variables are usually the physical parameters of an instrument, such as blow pressure, bow velocity, etc. In this context, the mapping of gestures to the synthesis inputs seems to be more evident, since the relation of these inputs to the synthesis algorithm are directly mapped by the multiple dependencies based on the physics of the particular instrument.

8.3.3.1 Mapping for General Musical Performance

Although simple one-to-one or direct mappings are by far the most commonly used, other mapping strategies can be used. For instance, through the use of several mapping strategies, it has been shown that for the same gestural controller and synthesis algorithm, the choice of mapping strategy became the determinant factor concerning the expressivity of the instrument.

The definition of mapping strategies using instrument-like and perhaps instrument-inspired controllers can benefit from our knowledge of the physics of acoustic instrument. But in the case of an alternate controllers, the possible mapping strategies to be applied are far from obvious, since no model of the mappings strategies to be used is available. Even so, it can be demonstrated that the choice of mappings influences user performance for the manipulation of general input devices in a musical context.

8.3.3.2 Mapping as Multiple Layer

Mapping can be implemented as a single layer between controller outputs and synthesis inputs. In this case, a change of either the gestural controller or synthesis algorithm would imply the definition of a different mapping.

One way to overcome this situation is the definition of mapping as two (or possibly more) independent layers: a mapping of control variables to intermediate parameters and a mapping of intermediate parameters to synthesis variables.

This means that the use of different gestural controllers would necessitate the use of different mappings in the first layer, but the second layer, between intermediate parameters and synthesis parameters, would remain unchanged. Conversely, changing the synthesis method involves the adaptation of the second layer, considering that the same abstract parameters can be used, but does not interfere with the first layer, therefore being transparent to the performer.

The definition of those intermediate parameters or an intermediate abstract parameter layer can be based on perceptual variables such as timbre, loudness, and pitch, but can be based on other perceptual characteristics of sounds, or have no relationship to perception, being then arbitrarily chosen by the composer or performer.

8.4 Commented bibliography

A good tutorial on Hidden Markov Models is [2]. Hiller and Isaacson [1] were the first to implement Markov chains in a musical application. The application of HMM representation of musical theme for search, described in Sect. 8.1.3.1, is presented in [3].

Bibliography

[1] Lejaren A. Hiller and L. M. Isaacson. *Experimental Music-Composition with an Electronic Computer*. McGraw-Hill, 1959.

[2] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceeedings of the IEEE*, 77(2):257–286, 1989.

[3] J. Shifrin, B. Pardo, C. Meek, and W. Birmingham. Hmm-based musical query retrieval. In *Proc. ACM/IEEE Joint Conference on Digital Libraries*, pages 295–300, 2002.

[4] Iannis Xenakis. *Formalized Music*. Indiana University Press, 1971.

Contents