

Probabilistic-logical Modeling of Music

Jon Sneyers, Joost Vennekens, Danny De Schreye

{jon,joost,dannyd}@cs.kuleuven.be

Department of Computer Science, K.U.Leuven,
Celestijnenlaan 200A, 3001 Heverlee, Belgium.

Tel: +3216327552 Fax: +3216327996

Abstract. PRISM is a probabilistic-logical programming language based on Prolog. We present a PRISM-implementation of a general model for polyphonic music, based on Hidden Markov Models. Its probability parameters are automatically learned by running the built-in EM-algorithm of PRISM on training examples. We show how the model can be used as a classifier for music that guesses the composer of unknown fragments of music. Then we use it to automatically compose new music.

Keywords: PRISM, probabilistic-logical programming, music classification, automatic music composition

1 Introduction

Music composers are bound to certain - mostly unwritten - rules, defining the musical genre of their work. In this paper we will construct a general model to describe such rules.

Many formalisms have been proposed to express strict, logical rules. A well-known example is the logic programming language Prolog. However, most musical genres are hard or impossible to describe as a set of strict rules. It seems to be inherent to music to be somewhat ‘random’ and ‘organic’.

Because of this inherent randomness, we will work in a probabilistic context, in which we represent musical rules as probabilistic experiments with some unknown probability distribution. We use a variant of Hidden Markov Models, for which the probability distributions can be computed in an automated way from a set of examples.

Probabilistic-logical programming is an extension of logic programming which allows programmers to express both statistical and relational knowledge in a natural way. We will use this formalism to build our model.

Traditionally, two types of models for music are distinguished. *Synthetic* models are used to generate music (automatic composition). *Analytic* models are designed to analyze (e.g. to classify) music. Conklin [2] pointed out that a general model can be applied to both tasks. A statistical analytic model can (in principle) be sampled to generate music.

The goal of this paper is to provide empirical evidence for the feasibility of implementing such a dual-use model, while showing the expressiveness and power of probabilistic-logical formalisms.

The paper is organized as follows. In section 2 we introduce a new representation for music that we use in the application. Section 3 recalls the basics of the programming language PRISM and shows how to model Hidden Markov Models in this language. In Section 4 we describe the system and the top-level of its implementation in PRISM. We report on our experiments with the system in Section 5 and conclude in Section 6.

2 Music Representation

For simplicity, we will consider only three aspects of music: melody (note pitch and octave), rhythm (note duration) and polyphony (different voices sounding together). We will ignore aspects like volume, timbre of instruments, articulations like the accent, staccato, portato, legato, . . .

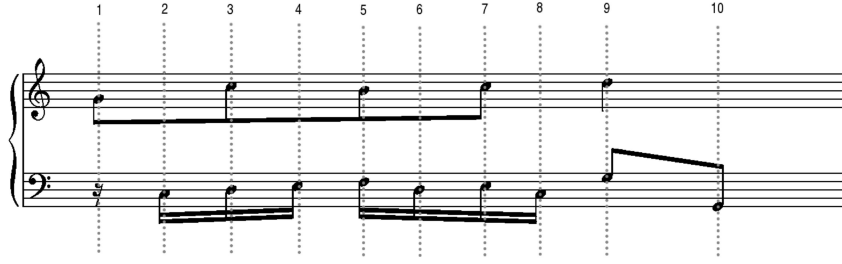
To represent a music score in a form which is suitable for both Prolog-like environments and Markov-like models, we will use the following notation, which we call the *Interwoven Voices List* (IVL) notation:

Definition 1. *An IVL of length n with v voices is a Prolog list term of the form*

$$[(D_0, V_0, N_0), (D_1, V_1, N_1), \dots, (D_n, V_n, N_n)]$$

where D_i are positive integer numbers and V_i and N_i are Prolog lists of length v . The elements of V_i are atoms: either the atom `new` or the atom `old`. The elements of N_i are either a positive integer number or the atom `r`. If for some i ($< n$) and j ($\leq v$), the j^{th} element of N_i and N_{i+1} are different, then the j^{th} element of V_{i+1} must be `new`. If for some i ($< n$) and j ($\leq v$), the j^{th} element of both N_i and N_{i+1} is `r`, then the j^{th} element of V_{i+1} must be `old`.

We say an IVL of length n consists of n phases. D_i is the duration of the i^{th} phase. The list N_i represents the notes that are played in every voice in the i^{th} phase. If no note is played by some voice, we use the symbol `r` (rest). We assume every voice to play only one note at the same time – if not, we can split them into as many voices as necessary. Note numbers encode both the pitch (c, cis=des, d, . . . , a, ais=bes, b) and the octave of a note. For example, c4 is represented as 48, cis4 as 49, d4 as 50, c5 as 60. When in some voice a note is followed by the same note, there are two possibilities: two distinct notes are played, or one long note is played (covering more than one phase because of shorter notes in the other voices). The list V_i indicates for every voice whether a new note is played (`new`) or the note played in the previous phase is continued (`old`). Figure 1 illustrates the representation on a small example.



[(16, [new,new], [55,r]), (16, [old,new], [55,36]),
 (16, [new,new], [60,38]), (16, [old,new], [60,40]),
 (16, [new,new], [59,41]), (16, [old,new], [59,38]),
 (16, [new,new], [60,40]), (16, [old,new], [60,36]),
 (32, [new,new], [62,43]), (32, [old,new], [62,31])]

Fig. 1. The Interwoven Voice List (IVL) notation

3 PRISM and HMM's

In this section we will briefly introduce the PRISM programming language [9, 10] and how to use it for Hidden Markov Models [1].

A PRISM-program consists of a logical and a probabilistic part. The probabilistic part defines a probability distribution $P_{\mathcal{F}}$ on a set \mathcal{F} of collections of ground facts. The logical part takes the form of a definite Horn clause program D , which serves to extend this distribution $P_{\mathcal{F}}$ to a distribution on the set of Herbrand interpretations of the program. More precisely, the probability of an interpretation I is defined as the sum of all probabilities $P_{\mathcal{F}}(F)$ of sets of facts $F \in \mathcal{F}$, for which the least Herbrand model of $(F \cup D)$ equals I .

The facts that can appear in such a set F are all of the special form $\text{msw}(\mathbf{s}, \mathbf{i}, \mathbf{v})$, with \mathbf{s} , \mathbf{i} , and \mathbf{v} ground terms. A term \mathbf{s} in such an atom is called a **multivalued switch** and it represents a random variable, which can take on a value from a certain domain. A declaration $\text{values}(\mathbf{t}, [v_1, \dots, v_n])$, with \mathbf{t} a term and all v_i ground terms, can be used to specify that the domain $\text{dom}(s)$ of \mathbf{s} is $\langle v_1, \dots, v_n \rangle$, for all switches \mathbf{s} which are ground instantiations of the term \mathbf{t} . For each switch \mathbf{s} and each value \mathbf{v} from $\text{dom}(s)$, the probability $P_s(\mathbf{v})$ of \mathbf{s} taking on this particular value needs to be defined. This is done using a query $\text{set_sw}(\mathbf{t}, [\alpha_1, \dots, \alpha_n])$, with \mathbf{t} a term, the α_i real numbers and n the number of values in $\text{dom}(s)$. The meaning of this is that $P_s(v_i) = \alpha_i$, with $\text{dom}(s) = \langle v_1, \dots, v_n \rangle$, for each switch \mathbf{s} which is a ground instantiation of \mathbf{t} .

Each set $F \in \mathcal{F}$ is of the form $\{\text{msw}(\mathbf{s}, \mathbf{i}, v_s^i) \mid \mathbf{s} \text{ is a switch, } \mathbf{i} \in I\}$, where I is some set of ground terms used to distinguish consecutive trials involving the same switch and each $v_s^i \in \text{dom}(s)$. All such trials are assumed to be probabilistically

independent, i.e., for every such set F :

$$P_{\mathcal{F}}(F) = \prod_{\text{msw}(\mathbf{s}, \mathbf{i}, \mathbf{v}) \in F} P_s(v).$$

Recently, an alternative syntax has been proposed, where instead of $\text{msw}(\mathbf{s}, \mathbf{i}, \mathbf{v})$ -atoms, $\text{msw}(\mathbf{s}, \mathbf{v})$ -atoms are now used. Programs in this new syntax should be read as though each occurrence of such an atom had an implicit extra argument \mathbf{i} which distinguishes it from all other occurrences of this predicate in the program.

The PRISM system has several useful properties. Firstly, the semantics of its language is easy to understand and quite natural. Secondly, the system offers an efficient learning algorithm, based on the well-known EM algorithm, which is able to estimate the probabilistic parameters $P_s(v)$, based on a set of observations. The built-in learning algorithm is provably as efficient as special purpose versions of the EM algorithm, such as the Baum-Welch algorithm for Hidden Markov Models, the Inside-Outside algorithm for Probabilistic context-free grammars, and the EM algorithm using evidence propagation for singly-connected Bayesian networks, provided that the models are appropriately programmed in PRISM.

For example, consider the experiment of flipping a coin a number of times. We could model this as follows:

```
target(flipN, 2).
values(coin, [heads,tails]).
flipN(0, []).
flipN(N, [T|Ts]) :- N>0, msw(coin,T), N1 is N-1, flipN(N1,Ts).
```

Let us say the coin is fair, i.e. both outcomes are equally probable. We can use the `set_sw` built-in to set the probability distribution of the coin experiment accordingly: `set_sw(coin, [0.5,0.5])`. Now we can use `sample(flipN(N,S))` to get a randomly generated sequence S of N coin flips.

Now let us assume we do not know the probability distribution of `coin` – maybe the coin has been tampered with – but we do have a sequence $S2$ of $N2$ coin flip outcomes. In this case, we can use `learn(flipN(N2,S2))` to estimate the probabilities. In this toy example, the result will simply correspond to the relative frequencies. In the learning phase, for every training example, all its *explanations* are enumerated, i.e. all combinations of `msw` outcomes that lead to the training example. In the coin flipping example there is just one explanation for every sequence of coin flips, but in general there can be many explanations for a single observation.

To illustrate how PRISM can be used to model HMMs, we consider the HMM in Figure 2. At each time point, this HMM probabilistically chooses both a successor state and an output symbol. Both these choices depend on the current state. To model this in PRISM, we need a switch $trans(s)$, representing the choice of a successor state, and a switch $out(s)$, representing the choice of an output symbol, for every state s .

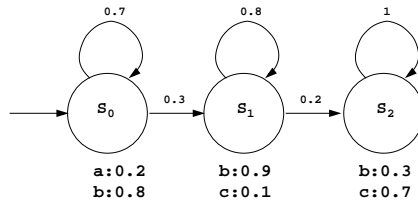


Fig. 2. A Hidden Markov Model

```

values(trans(s0), [s0,s1]).
values(trans(s1), [s1,s2]).
values(trans(s2), [s2]).
values(out(s0), [a,b]).
values(out(s1), [b,c]).
values(out(s2), [b,c]).
  
```

Now, the state of the HMM at time T can be defined as follows:

```

state(s0,0).
state(Next, T) :- T > 0, TPrev is T - 1,
  state(Prev, TPrev), msw(trans(Prev), Next).
  
```

The output at time T can be defined as:

```

out(Char, T) :- state(State, T), msw(out(State), Char).
  
```

We can now define a predicate $hmm(S, T)$ to express that string S is generated after T steps, by simply gathering all the produced symbols into a list:

```

hmm([], 0).
hmm([Char | Chars], Time) :-
  Time > 0, TimePrev is Time - 1,
  out(Char, TimePrev), hmm(Chars, TimePrev).
  
```

The probabilities for the various switches can be set as follows:

```

:- set_sw(trans(s1), [0.8,0.2]), set_sw(trans(s2), [1]),
  set_sw(trans(s0), [0.7,0.3]), set_sw(out(s0), [0.2,0.8]),
  set_sw(out(s1), [0.9,0.1]), set_sw(out(s2), [0.3,0.7]).
  
```

Alternatively, these values can also be estimated on the basis of a set of `hmm/1` observations, which are called *training examples*.

When the switch probabilities are set, the model can be sampled using the `sample/1` built-in. For example, the query `sample(hmm(L,10))` would result in unifying L to the length 10 list outputted by a random execution of the given HMM. The probability of an observed sequence can be computed using the `prob/2` built-in.

4 Modeling IVL-music in PRISM

Hidden Markov Models sequentially process data streams. For analytic music models, this approach makes sense: after all, this is how a human listener perceives music. However, if the model is also to be used as a synthetic music model, it forces a left-to-right composing strategy, which is not the way human composers usually work. Therefore, we will use *nested* HMMs, based on the intuition of global and local music structure. The global structure of a piece of music is captured by transitions between various Song States (SS). Within every such song state, there is one HMM for each voice, which captures the local structure for that particular voice by transitions between various Voice States (VS). This principle is illustrate in Figure 3.

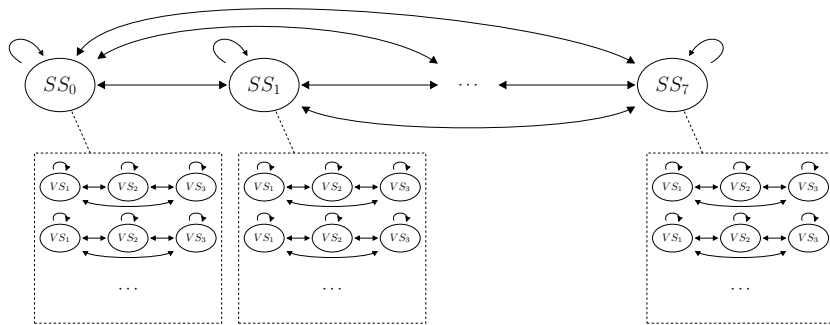


Fig. 3. Nested HMM structure

A music fragment is encoded in facts of the form `song(NV,L,BT,IVL)`, where IVL is an IVL of length L with NV voices. For all examples, we manually determined the base tone (BT) to allow the model to transpose the notes. Doing this, everything is learned modulo translations in pitch. All probabilities are learned by performing the PRISM built-in EM algorithm on a set of `song/4` facts.

We will now discuss the main predicates of the source code of our model. The core of the model is a nested Hidden Markov Model. Every transition into the next phase probabilistically selects one out of seven “song states”, the probabilities depending on the current song state. For every voice, one out of three “voice states” is chosen, the probabilities depending on the current song state and the current voice state for that voice. All voice output (pitches, octaves, `old/new` and rests) probabilities depend on the current song state and the current voice state for the relevant voice. Most song states restrict the domain for output pitches to some harmonically relevant subdomain. The phase duration probabilities depend only on the current song state.

Note that the number of song states s and the number of voice states v can be chosen arbitrarily. The number of probability parameters to learn grows linearly as sv grows, but the number of explanations (and hence the duration of the learning phase) for a single training example IVL of length l with n voices

roughly corresponds to $(sv^n)^l$. Since the training fragments have to be long enough to exhibit some musically meaningful properties, we are forced to use small values for the other variables. We used the values $s = 7$ and $v = 3$, and for training $n = 2$ and $l \approx 20$, because larger values quickly became computationally infeasible. However, they are probably too low to capture sufficient detail. The quality of the model can be improved by using more states, although it will also become more likely to obtain insufficiently generalizing parameter values in the training phase. The domains of the different probabilistic experiments are given below:

```

values(tr_ss(_PrevS),[1,2,3,4,5,6,7]). % song state transitions
values(tr_vs(_S,_PrevV),[a,b,c]). % voice state transitions
values(out_D(_S),[8,16,32,48,64,96,128]). % durations
values(out_V(_S,_V),[old,new]). % old or new note
values(out_rest(_S,_V),[rest,note]). % rest or real note
values(out_octave(_S,_V),[-2,-1,0,1,2]). % relative octave
values(out_modnote(1,_V),[0, 4, 7]). % c e g
values(out_modnote(2,_V),[0, 2, 4, 5, 7, 9, 11]). % c d e f g a b
values(out_modnote(3,_V),[0, 2, 4, 6, 7, 9, 11]). % c d e fis g a b
values(out_modnote(4,_V),[0, 2, 4, 5, 7, 9, 10]). % c d e f g a bes
values(out_modnote(5,_V),[0, 3, 4, 5, 7, 8, 10]). % c es f g as bes
values(out_modnote(6,_V),[0, 2, 3, 5, 9, 10]). % c d es f a bes
values(out_modnote(7,_V),[0,1,2,3,4,5,6,7,8,9,10,11]). % all pitches

```

The main loop of the program is a nested HMM: after every output observation, a state transition is made.

```

song(NV,L,BT,IVL):-
    initlist(NV,4,0_prev), initlist(NV,a,VoiceState),
    hmm(NV,1,L,1,VoiceState,BT,IVL,_,0_prev).

% length L; NV voices; base tone BT; current phase (D,V,N)
hmm(NV,T,L,SongState,VoiceState,BT,[(D,V,N)|Y],N_prev,0_prev) :-
    T < L, T > 1, T1 is T+1,
    observe(NV,SongState,VoiceState,BT,D,V,N,N_prev,0_prev),
    tr_voicestates(NV,SongState,VoiceState,VS_Next),
    msw(tr_ss(SongState),SS_Next),
    octaves(BT,N,0_prev,N_octave),
    hmm(NV,T1,L,SS_Next,VS_Next,BT,Y,N,N_octave).

```

The `observe/9` predicate checks one IVL phase (D, V, N) . First the phase length D is checked. Then the list elements of V and N are checked one by one.

```

observe(NV,SongState,VoiceState,BT,D,V,N,N_prev,0_prev) :-
    msw(out_D(SongState),D),
    nlist(NV,SongState,VoiceState,BT,N,V,N_prev,0_prev).

```

For every note, it is first determined whether it is a rest or a “real” note. The note value of the “real” note is checked in `check_real_note`. When it is identical to the previous note, we check whether it is new or old. Note that these experiments depend on both the song state (which is shared by all voices in given IVL phase) and the voice state of the corresponding voice.

```
nlist(NV,SongState,[VoiceState|RVS],BT,[N|RN],V,[PN|RPN],PrO) :-
    NV > 0, NV1 is NV-1, V=[ON|RON], PrO=[PO|RPO],
    check_note(SongState,VoiceState,BT,N,PO), % check note itself
    check_new(SongState,VoiceState,N,PN,ON), % check old/new
    nlist(NV1,SongState,RVS,BT,RN,RPN,RON,RPO).
nlist(0,_, [],_ , [], [], [], []).
```

```
check_note(SongState,VoiceState,BT>Note,PrevOct) :-
    msw(out_rest(SongState,VoiceState),X),
    check_note(SongState,VoiceState,BT>Note,PrevOct,X).
```

```
check_note(_,_ ,_,r,_,rest).
check_note(SongState,VoiceState,BT>Note,PrevOct,note) :-
    check_real_note(SongState,VoiceState,BT>Note,PrevOct).
```

```
check_new(SongState,VoiceState,X,X,OldNew) :-
    number(X), msw(out_V(SongState,VoiceState),OldNew).
check_new(_,_ ,r,r,old).
check_new(_,_ ,A,B,new) :- A \= B.
```

In `check_real_note`, we check the pitch and the relative octave, i.e. the difference compared to the octave of the previous note (of that voice).

Note that thus far, the program can be used both for analysis and generation, i.e. we can call the `song(NV,L,BT,IVL)` predicate with instantiated or partially uninstantiated arguments. However, when we are checking the note values, we will need two clauses for `check_real_note`, one for analysis/training and one for synthesis/sampling. When sampling, `sanity_check` is called to avoid too high or too low notes: it enforces the octave to stay in the range 2-6.

```
check_real_note(SongState,VoiceState,BT>Note,PrevOct) :-
    number(Note), % training
    Pitch is (Note-BT) mod 12,
    OctDiff is ((Note-BT) // 12) - PrevOct,
    msw(out_modnote(SongState,VoiceState),Pitch),
    msw(out_octave(SongState,VoiceState),OctDiff).
```

```
check_real_note(SongState,VoiceState,BT>Note,PrevOct) :-
    var(Note), % sampling
    msw(out_modnote(SongState,VoiceState),Pitch),
    msw(out_octave(SongState,VoiceState),OctDiff),
```

```
NewOctave is PrevOct + OctDiff,  
sanity_check(NewOctave,SaneNewOctave),  
Note is BT + Pitch + 12*SaneNewOctave.
```

5 Experimental results

5.1 Classification

Pollastri and Simoncelli [7] have used HMMs for classification of melodies by composer. In a similar way, we will use our model to classify fragments of polyphonic music. To keep things simple, we will consider only two composers: Bach and Mozart.

Using the built-in EM algorithm of PRISM, we train two instances of the model, M_B and M_M , the former using fragments from works of Bach, the latter using fragments of Mozart. To classify a new, unknown fragment we compute the probabilities of it being the output of M_B or M_M . The fragment is classified as being work of the composer for which the probability is highest.

The computational complexity of this method is dominated by the training phase – which took something in the order of tens of minutes on a Pentium 4 machine with 1 gigabyte of RAM. Computing the probability of an unknown fragment took only one or two seconds. This is an interesting property of this approach, since training the model has to be done only once for every classification category, after which any number of fragments can be classified in reasonable time.

The experiments were originally performed in PRISM 1.6. In the preparation for this paper, we repeated some experiments using PRISM 1.8.1, with comparable results. For training M_B , 72 fragments from Bachs *Inventions* (BWV 772,773,775,779) were used. We used 50 fragments from Mozarts *duets for horn* (KV 487, numbers 1, 2, 5, 8, 9, 10 and 12) to train M_M . Together, this amounts to 1428 IVL phases for M_B and 948 IVL phases for M_M . All training examples have two voices.

Figure 4(a) gives an overview of the classification results for 30 other fragments. The numbers P_B and P_M are the log-probabilities of the fragment being the output of M_B and M_M , respectively. If $P_B > P_M$, the fragment is classified as “composed by Bach”, otherwise it is classified as “composed by Mozart”. For all 30 fragments, the classification was correct. This is a remarkable result, since 20 of these fragments were chosen from totally different work compared to the fragments used for training the models, and they have three voices, not two like the training examples.

In the fragments we used, it has to be noted that the phase durations D_i are most often 16th notes in pieces of Bach, while in pieces of Mozart the 8th note is more prominent. As a result, the correct classification may be mostly based on the typical durations of the IVL phases. We want of course to find out whether the trained models (implicitly) contain more musical style properties than just the notational tempo. To check whether our classification method still

<i>id</i>	P_B	P_M	<i>guess:</i>	<i>reality</i>
b01	-129 >	-655	Bach	
b02	-119 >	-585	Bach	BMV 784
b03	-119 >	$-\infty$	Bach	(Bach)
b04	-116 >	$-\infty$	Bach	2 voices
b05	-121 >	$-\infty$	Bach	
m01	-297 <	-136	Mozart	KV 487
m02	-124 <	-114	Mozart	number 7
m03	-302 <	-120	Mozart	(Mozart)
m04	-203 <	-155	Mozart	2 voices
m05	-152 <	-116	Mozart	
b06	-144 >	-236	Bach	
b07	-189 >	-422	Bach	
b08	-178 >	-216	Bach	BMV 798
b09	-158 >	-317	Bach	(Bach)
b10	-156 >	-608	Bach	
b11	-166 >	-234	Bach	
b12	-174 >	-492	Bach	
b13	-161 >	-166	Bach	3 voices
b14	-153 >	-464	Bach	
b15	-171 >	-176	Bach	
m06	$-\infty$ <	-553	Mozart	
m07	$-\infty$ <	-230	Mozart	
m08	-431 <	-234	Mozart	KV 229
m09	-283 <	-165	Mozart	
m10	-424 <	-189	Mozart	(Mozart)
m11	-324 <	-119	Mozart	
m12	-429 <	-175	Mozart	
m13	-397 <	-169	Mozart	3 voices
m14	-330 <	-179	Mozart	
m15	-381 <	-175	Mozart	

(a) all information

<i>id</i>	P_B	P_M	<i>guess:</i>	<i>reality</i>
b01	-132 >	-140	Bach	
b02	-118 >	-125	Bach	BMV 784
b03	-121 >	-142	Bach	(Bach)
b04	-122 >	-140	Bach	2 voices
b05	-121 >	-143	Bach	
m01	-110 <	-103	Mozart	KV 487
m02	-100 >	-102	Bach	number 7
m03	-120 <	-110	Mozart	(Mozart)
m04	-102 <	-100	Mozart	2 voices
m05	-99 >	-101	Bach	
b06	-144 >	-162	Bach	
b07	-152 >	-158	Bach	
b08	-161 >	-171	Bach	BMV 798
b09	-162 >	-168	Bach	(Bach)
b10	-155 >	-156	Bach	
b11	-165 >	-169	Bach	
b12	-168 >	-172	Bach	
b13	-156 >	-167	Bach	3 voices
b14	-156 >	-166	Bach	
b15	-147 >	-154	Bach	
m06	-256 <	-178	Mozart	
m07	-177 <	-149	Mozart	
m08	-237 <	-167	Mozart	KV 229
m09	-153.7 <	-153.6	Mozart	
m10	-179 >	-182	Bach	(Mozart)
m11	-162 <	-109	Mozart	
m12	-174 <	-168	Mozart	
m13	-187 <	-155	Mozart	3 voices
m14	-156 >	-169	Bach	
m15	-185 <	-167	Mozart	

(b) ignoring durations

Fig. 4. Classification results

works without the strong duration differences, we could preprocess the training and testing data, scaling it to a common tempo. Instead, we repeated the experiment, adapting our model so that it does not consider the phase duration information at all. By not considering the durations, only the melody and polyphony information can be used to classify the fragments. In figure 4(b), the results are given for this new classification method, which does not take into account the phase durations. Of the 30 fragments, 26 were classified correctly. This is of course significantly better than random guessing.

5.2 Music generation

We have also used our model to generate music: first we have trained the model using one of the training sets, then we sample the model using the `sample/1` built-in. Input parameters are the first notes, the base tone and the length (number of IVL phases). We arbitrarily chose (respectively) `c3` and `e4` of duration one eighth, base tone 0 (`c`), and 30 phases. Two typical examples of the resulting output are given in Figure 5, and can be downloaded at [11].

Although the output does not even come close to a human-composed piece, it seems to contain musical style elements of the training examples. It cannot directly be used to generate acceptable full songs, but it might be an interesting source of inspiration for human composers.

We expect that using more refined models (e.g. with more HMM states and adding the concept of time signature to avoid excessive syncopation) and using larger sets of training examples, the style of the generated music would be increasingly indistinguishable from the style of the original training examples.

6 Conclusion

We have presented a simple, yet general model for music. It can handle any number of voices. The probability parameters can automatically be adjusted for any musical genre given sufficiently many training examples of the genre.

The classification method presented in section 5.1 seems to be a promising approach, given the outcome of the experiments. Experiments on a larger scale, involving more composers and larger datasets, have to be performed to get an accurate idea of the scope of its practical applicability.

In its current form, the model described in this paper cannot be used for fully automatic music composition. However, its output might be an interesting source of inspiration for human composers. In particular, in the context of computer assisted composition, our system could perform a very useful role by suggesting fragments or polyphones to the composer.

The main contribution of our work, however, lies in demonstrating the feasibility of using probabilistic-logical programming as an elegant tool for developing applications in the computer music domain. As we argued in the introduction, two aspects seem important in modelling music applications: logical rules (or, alternatively, constraints) and probabilities. It is interesting to observe that these



(a) Trained with fragments of Bach



(b) Trained with fragments of Mozart

Fig. 5. Sample of the model trained with fragments of Bach and Mozart

two aspects have been considered separately by different researchers in the past as a basis for developing music systems.

Constraints and constraint processing have been successfully applied in a number of applications (see e.g. [12, 8, 4] and related work at IRCAM). The approach is natural, since music is governed by many constraints. However, there are even more 'weak' constraints involved in the application. These are more naturally expressed by probabilistic rules or HMMs. Another group of researchers has focussed on probabilities and HMMs as a basis for developing music analysis and synthesis applications (e.g. [7, 6]). An important disadvantage of that approach is that it is not very flexible: it is hard to incorporate expert knowledge in such models or to experiment with variants of HMMs.

Our work combines these two aspects. In this respect, it is somewhat similar to the work of D. Cope [3], whose Experiments in Musical Intelligence system intelligently recombines randomly chosen parts of existing work to satisfy certain musical constraints. However, in contrast to this system, we have chosen to follow a declarative approach, in which we describe a general model of a piece of music, rather than develop specific algorithms to derive new music from old.

As far as we know, our work is the first application of a declarative language based on probabilistic logic to this problem. Our experiments show that the development of the applications in such a language is particularly elegant and that it provides a functionality of analysis and synthesis using the same model

(and program). In developing the application, it feels as if PRISM was designed for this purpose.

Future work. An important musical concept, ignored in our model, is the meter, which distinguishes the stronger and more important notes from the decorative intermediate notes. One approach to incorporate this concept in the model is to change the representation: instead of using one flat list, a list of lists could be used where each sublist contains one bar. The model could then be refined, e.g. by handling the first note of every bar in a different way or by adding a higher level HMM state which has a transition after every bar instead of after every phase. It might also be useful to abandon the rather naive representation of a piece of music as a list of notes and move to a more structured representation such as, e.g., that used in [5].

As mentioned before, the results presented here should be validated with a more thorough experimental evaluation. The main difficulty will be to collect and prepare a large dataset.

References

1. Yoshua Bengio. Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162, 1999.
2. Darell Conklin. Music Generation from Statistical Models. In *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, pages 30–35, Aberystwyth, Wales, 2003.
3. David Cope. *The Algorithmic Composer*. Madison, WI: A-R Editions, 2000.
4. Martin Henz, Stefan Lauer, and Detlev Zimmermann. COMPOzE—Intention-based Music Composition through Constraint Programming. In *ICTAI '96: Proceedings of the 8th International Conference on Tools with Artificial Intelligence (ICTAI '96)*, page 118, Washington, DC, USA, 1996. IEEE Computer Society.
5. Paul Hudak, Tom Makucevich, Syam Gadde, and Bo Whong. Haskore music notation - an algebra of music. *Journal of Functional Programming*, 6(3):465–483, 1996.
6. Yuval Marom. Improvising Jazz using Markov chains. Honours Thesis, University of Western Australia, 1997.
7. Emanuele Pollastri and Giuliano Simoncelli. Classification of Melodies by Composer with Hidden Markov Models. In *Proceedings of the First International Conference on WEB Delivering of Music*, pages 88–95, Firenze, Italy, 2001.
8. Camilo Rueda and Frank D. Valencia. Situation: A Constraint-Based Visual System for Musical Compositions. In Carlos Agon, editor, *The OM's Composer Book*, IRCAM Centre Pompidou, France. To appear.
9. Taisuke Sato and Yoshitaka Kameya. PRISM: A symbolic-statistical modeling language. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1330–1335, Nagoya, Japan, 1997.
10. Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research (JAIR)*, 15:391–454, 2001.
11. Jon Sneyers. MIDI files of automatically generated music. Available at http://www.cs.kuleuven.be/~jon/automatic_composition/.

12. Charlotte Truchet, Gérard Assayag, and Philippe Codognet. Visual and Adaptive Constraint Programming in Music. In *International Computer Music Conference (ICMC01)*, La Havana, Cuba, September 2001.