Performance Rendering using Structure Level Expression

Bastiaan J. van der Weij
5922151

Bachelor thesis
Credits: 15 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Prof. dr. ir. R.J.H. Scha

Institute for Language and Logic
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 24th, 2010

Abstract

Both machine learning and rule based techniques have been extensively applied to performance rendering. However, relatively few systems make explicit use of machine learning combined with musical structure. Systems that use machine learning usually learn expression at note level. This paper introduces a performance rendering system that learns expression exclusively at a structural level. The system can be seen as complementary to systems that learn expression at note level.

Keywords: performance rendering, musical structure, constituent structure, polyphonic piano music

1 Introduction

In the Western art music tradition it is customary that composers write down their compositions as scores. The task of a performer is to some extent to accurately reproduce the score; however, a perfect reproduction of a score generally sounds robotic and unpleasant. What makes a performance appealing is that the performer deviates from the score, altering timing, articulation and loudness, creating an expressive performance.

Given a hypothetical perfect synthesizer, performing scores with computers is a trivial task. However *expressively* performing music is not and much research has focussed on this issue. A wide selection of scientific fields may have interest in such research: Artificial intelligence, cognitive science and musicology all benefit from a better understanding of expression in music.

A rather practical approach to expressive music performance is to recreate it. A performance rendering system is a computer program that automatically creates an expressive, humanlike performance of a musical score. Performance rendering systems have always faced a problem of evaluation. It is debatable how much we should value correlation of generated performances with human performances of the same piece. A Chopin interpretation of Arthur Rubinstein can be radically different from a Chopin interpretation of Vladimir Horowitz, yet both are considered high quality interpretations. This problem has been one of the motivations for a yearly performance rendering event where performance rendering systems compete and are judged by an audience.

Over the years, many performance rendering systems have been proposed. Early attempts, like Director Musices [2], have applied sets of rules to the problem. As the availablity of corpora grew, machine learning systems have been proposed as well. Gerhard Widmer is working with a large dataset containing almost all works by Chopin, performed by one pianist. A performance rendering system using this dataset called YQX [1] was proposed by Sebastian Flossman et al. in 2008. This system won in al three categories of Rencon2008. At Rencon2011 it was ranked best in a blind evaluation by experts and ranked second in the evaluation by an audience. It should be noted that despite its age, Director Musices ranked third in the blind evaluation, suggesting that although performance rendering systems have advanced, the field is still in its infancy.

Many of the performance rendering systems that use machine learning, including YQX, predict expression at note level. Some do use structure. Pop-E[4], a rule based performance rendering system, uses a structure analysis based on Lerdahls and Jackendoff's grouping preference rules, but the structural analysis must be supplied to it by a human expert. In this thesis we will motivate the need for structure level expression and propose a system that purely uses structure level expression. The YQX system is very successful at the moment, therefore we will use this system as a reference for introducing our own system. In the following section we will provide a short analysis of some shortcomings of the YQX system and argue what we believe is the reason for these shortcomings.

What we mean by structure level expression and why we think it is important is clarified in section 2. Section 3 will describe the system as a whole. Section 4 will describe every component of the system in detail. Practical issues are discussed in section 5. Results are presented in section 6 and discussed in section 7. Suggestions for improving the system and future research can be found in section 9.

2 Musical Structure

2.1 Motivation

The YQX system, as Flossman et al. admit, tended to sometimes produce nervous sounding changes in expression. They present two extensions tailored towards generating smoother performances. The problem with these extensions is, as Flossman et al. also admit himself, is that the increased smoothness comes at the expense of expressitivity. To compensate for this, three explicit rules are added to postprocess the performances. We think the reason that Flossman et al. stumbled upon this tradeoff between expressiveness and nervousness is that the nervousness is inherent to note level performance rendering. Note level systems simply are not suited to capture one component of expression, namely structure level expression.

The system presented here will use structure exclusively to generate performances and will completely ignore note level expression. We hope that this system is able capture the sudden changes of expression that a note level system cannot easily learn. Sudden changes in expression occur frequently in expressive performances, a few notes may be played very loud, or very slow followed by soft notes played very fast. When done at the right moment, such sudden changes can greatly improve the humanlike feel of the performance, however when done at the wrong moment, they can ruin a performance. We think that at structure level it will be easier to capture the context in which sudden changes occur.

2.2 Structure Level Expression

When listening to music, the listener's musical intuition assigns a certain hierarchical structure to the music: Notes make up phrases, phrases make up themes and themes make up a piece. In a performance, this structure may be accentuated in different ways. Accentuation happens at different levels, at note level performers may slow down at the end a phrase or introduce small pauses in the performance at phrase transitions. At constituent level one phrase may be played very loud, fast or staccato, while the next may be played slow, soft and legato.

To formally describe musical structure, we can look at music in a way similar to the way we look at natural language processing(NLP). In this analogy we see a piece of music as a sentence, which consists of constituents that individually can be made of constituents as well. We can recursively subdivide constituents in more constituents until we reach a terminal symbol. In NLP this may be a word, in music, this may be a note. We could represent musical structure as a parse tree. This paradigm corresponds to the intuition that a melody is not simply a sequence of notes but that notes form phrases. A phrase is always a constituent but a constituent is not always a phrase, it can be a distinguished part of a phrase or a distinguished set of phrases as well.

We must note that musical scores can be highly ambiguous and even experienced listeners may not agree on the correct parsetree of a piece. Quite often there may simply be more than one parse tree that makes musical sense. This should not be a problem for a performance rendering system: different expressive interpretations of one piece can be very diverse and still be accepted as sensible interpretations of the piece. As long as the parse tree does make at least some musical sense, a performance rendering system should be able to use it.

Although the YQX does have some notion of structure[1], expression is only predicted per note. The authors admit that the first simple version of the system "tended to sometimes produce unstable, 'nervous' sounding performances". The only way to overcome this problem was to introduce methods that limited the expressiveness of performances. We consider this trade-off to be inherent to note-level expression based systems. To solve it, some notion of structure level expression is required.

3 Approach

In this thesis, we propose a structure based performance rendering (SBPR) system. The system presented here ignores note level expression. Instead we will try to predict only constituent (or structure) level expression. The assumption is that this kind of expression really exists in performances and that is different and independent from note level structure. We think that a constituent level system also corresponds better to how actual human performers play music.

The system will be similar to YQX in a number of ways, but with the crucial difference that expression will not be predicted per note, but per constituent. Every constituent will be played with consistend expression, the articulation, dynamics and tempo change only at constituent breaks.

We use a corpus that contains performances and corresponding scores of Western classical piano music. Every note in every performance has been associated with the corresponding score note so we can see exactly how every note in the score was played. The performances are of high quality and played by famous pianists. See section 5.1 for more details on the corpus.

A structural analysis is used to derive a hierarchical structure for every score in the corpus, however, to keep the system simple we will only use this structural analysis to create a segmentation of the score into constituents. After segmentation, four score features, two of which are direct generalizations of YQX's score features, are extracted for each constituent.

So far, we have only used the score. Since we have a segmentation and every

[1]One of the note features is distance to nearest point of closure

score note is associated with a performance note[2] we can also define expression per constituent. Three parameters, analogous to YQX's targets, will be used to describe expression per constituent.

The segmentation, score features and expression parameters are based only on the *melody* of the piece. In this case, melody is defined to be the highest notes in the top staff.

The resulting data is used to train a first order hidden Markov model. The system uses this model to generate performances given a score. To do this, the score is segmented into constituents, score features are extracted for each constituent. Finally viterbi decoding is used to find the sequence of expression parameters that best explains the observed score features.

The succes of a SBPR system depends largely on two factors. The ability to generate musically meaningful parse trees of a piece and the ability to accurately characterize the individual constituents and their relations with other constituents in score features. The following section address these issues.

4 Method

This section will describe individual components of the system sketched in section 3 in more detail. The structural analysis is based on the delta framework, which will be described in section 4.1. Section 4.2 will discuss how the delta framework is applied to get a segmentation. Sections 4.3 and 4.4 will describe how the scorefeatures and expression parameters are calculated. Section 4.5 will describe how a hidden Markov model is trained on our data.

4.1 The Delta Framework

In his Phd thesis [7], Markwin van der Berg introduces a formal way of parsing music into parsetrees: the delta framework. He relates his work to the work of Lerdahl and Jackendoff [6] but claims to have found a more general approach. Below, I will shortly describe the delta framework as proposed by Van der Berg.

The delta framework is based on the intuition that differences between features of notes indicate splits between constituents. The higher the difference, the higher the level of split in the parse tree (where the root note is at the highest level). Van der Berg proposes a delta rule that converts a set of differences, or deltas, between notes into a parsetree following this intuition.

The differences between notes are defined as the difference in value of a certain note feature. More formally, we can look at a piece of music as a sequence of notes, ordered by onset time:

$$M_{ij} = [n_i, n_{i+1}, \cdots, n_j]$$

A set of basic features, Φ, is assigned to each note. These are: **onset**, **pitch**, **loudness** and **release** (called offset by Van den Berg). From these, two other features can be derived: **duration** and **virtual duration**. Duration is defined as **release**(n_i) - **onset**(n_i) while virtual duration is defined as **onset**(n_{i+1}) - **onset**(n_i).

[2]In reality, not every score note is associated with a performance note since the pianist may have missed some notes. These notes will be ignored

The basic note features can be used to define delta functions, for example $\mathbf{\Delta Pitch} = \mathbf{Pitch}(n_i) - \mathbf{Pitch}(n_{i-1})$. In general, a delta function $\delta(i)$ is defined as the difference of two notes in some feature ϕ: $\delta(i) = \phi(n_i) - \phi(n_{i-1})$. We can apply a delta function to every pair of succeeding notes in a sequence to get a list of deltas:

$$\Delta M_{ij} = [\delta(n_{i+1}), \delta(n_{i+2}), \cdots, \delta(n_j)]$$

A recursive rule, called the delta rule can be used to translate a list of deltas into a grouping structure. This ruled, called the delta rule is shown in algorithm 1, where DeltaRule(M_{ij}) is a recursive call to the algorithm itself, M_{ij} is the ordered list of notes to be analyzed and A is the resulting analysis.[3]:

Algorithm 1 The delta rule

```
$D_{i+1,j} \leftarrow \Delta M_{ij}$
$A \leftarrow []$
$m \leftarrow$ max($D$)
for $\delta$ in $D$ do
  if $\delta = m$ then
    $p =$ index of $\delta$ in $D$
    $q =$ index of the next occurence of m in $D$ or $j$
    append DeltaRule($M_{pq}$) to $A$
  end if
end for
return $A$
```

The delta rule converts a sequence of notes into a nested list structure that can be interpreted as a tree. The deltarule can ‘parse’ a piece of music into a *parse tree.*

It is also possible to define higher order delta function (deltas of deltas). Second order deltafunction can be used to differentiate a group of ascending notes from a group of notes that have wildly varying pitches. Van den Berg notes that this generates ambigous grouping structures as for example a second order deltarule needs at least three notes to be specified. For three notes, three different grouping structures are possible: $[[n_1, n_2], n_3]$, $[n_1, n_2, n_3]]$ and $[n_1, [n_2, n_3]]$. Because we want to use a second order deltafunction for segmentation only, we choose to use only one interpretation: $[[n_1, n_2], n_3]$. This choice is motivated by the intuition that if in a group of three notes the delta function in some feature between the first two notes is low and the delta function between the second and third note is high (thus resulting in a high second order deltafunction) we would want a constituent split to happen between the second and third note. Note again that segmentation, like structure, may often be ambigous and multiple interpretations may be right; for our purposes it is important that at least one of these right interpretations is chosen.

Having defined first and second order deltafunctions and the deltarule, we can use first and second order deltafunctions to parse a piece of music into a parse tree. Since we have a set of six basic note features, we have a set of six interpretations(parse trees) available for a piece of music. Intuition tells us that if a particular group of notes is found in multiple parse trees, this group of

[3]The version here is a slightly reformulated, although functionally equivalent, version of Van den Berg's deltarule. See [7] for his original version.

notes may be eligible to be a constituent. Van den Berg captures this intuition in the form of a *yield rule*. A node in the parse tree is said to 'yield' a group of notes if the node recursively contains this group of notes. If two or more nodes in different trees yield the same group of notes they should be connected according to the yield rule.

Unfortunately, the yield rule addresses the problem of how to interpret multiple parse trees quite poorly. We would like to have some way of combining parsetrees into one 'consensus tree'. This is not what the yield rule does. The yield rule generates a set of trees in which some nodes may be interconnected but there does not seem to be a logical way to interpret this set of interpretations and connected nodes. Van den Berg does not further address this issue. In the next section we circumvent the problem completely by not using more than one interpretation at a time.

4.2 Segmentation

A recursive structure like the parse trees generated by the delta framework is the kind of structure that we would ideally want to attribute to music. However, because of the problems with interpreting different parsetrees and because deltatrees do not necessarily represent the kind of structure that human musical intuition would attribute to music, we chose not to use a hierarchical structure and instead settle for a segmentation, which can be more reliably derrived from deltatrees.

The goal is to find a 'safe' method to segment music, that is, a method that generates segmentations that do not clash with human musical intuition. For this purpose the parse trees of delta onset (inter-onset interval, or simply IOI) and delta pitch (pitch interval) seem most suited: sudden jumps in pitch or onset often correspond to phrase transitions.

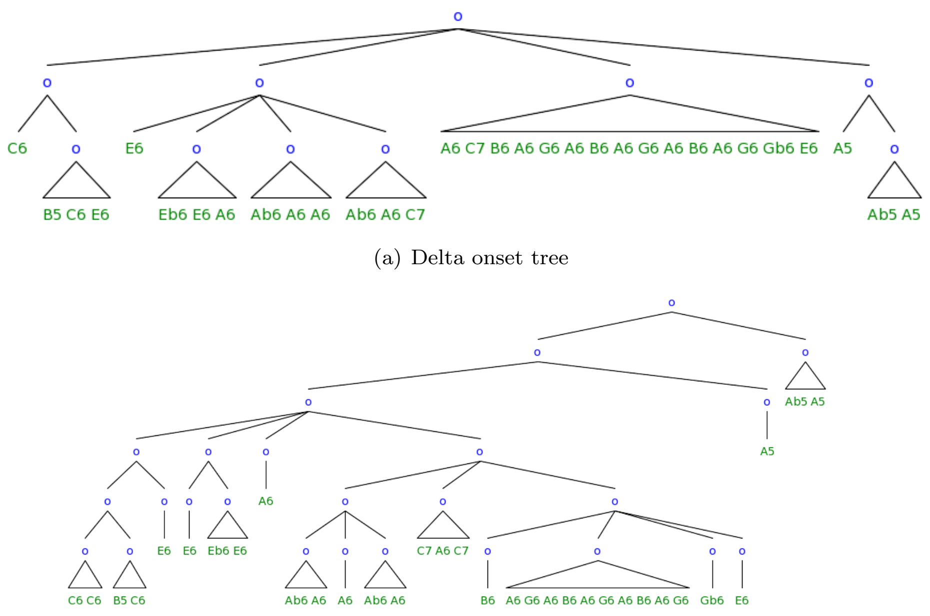
A delta onset tree and a second order delta pitch tree of the first eight bars of Mozart's Turkish March can be found in figure 1. In his figure we can see that some nodes contain only subnodes, some nodes contain only notes and some nodes contain notes and subnodes. To translate the tree into a segmentation we use algorithm 2 starting with the root node.

Algorithm 2 Segmentation

For some node n, do:

1. If n is a note, add it as a singleton constituent to the segmentation
2. Expand n.
3. If n contains only subnodes, or less than x notes, start from step 1 for every subnode and -note.
4. If n contains at least x notes, add the notes recursively contained by the current note to the segmentation.

The value of x represents the tolerance of singleton constituents. We need this parameter because sometimes very long notes cause top level splits in the onset tree. We have found that setting it to the number of notes recursively



(a) Delta onset tree

(b) Second order delta pitch tree

Figure 1: Parse trees of the first eight bars of piano sonata KV331 III. (Turkisch March)by Mozart

contained (yielded) by the current node divided by 16 works quite well for works by Mozart. Our segmentation of the first eight bars of Mozart's Turkish March using a delta onset tree can be found in figure 2.[4]

For some music like Mozart's, using only a delta onset tree works very well. However, some music, like Bach's Inventionen uses almost no differentiation in IOI. Hence, a segmentation based solely on inter onset interval will not work well for these pieces.

To overcome this problem we will use pitch. While Bach's Inventionen use almost no differentiation in IOI they can be segmented nonetheless. We will not make any attempt to combine pitch interval trees and IOI trees. Instead we take the segmentation given by the IOI tree and try to subdivide every segment using a pitch interval tree.

The sort of changes in pitch that indicate constituent transitions are not characterized very well by a first order pitch interval tree. For example a melody that has four notes that are chromatically ascending followed by four notes that have octave intervals between them should be segmented into two constituents where one contains the four ascending notes and another one contains the notes with octave intervals. A first order pitch interval tree puts the four ascending notes in one segment but puts the four octave interval notes in four separate segments. A second order pitch interval puts the split in this case exactly where we would want it.

[4]Note that if the delta tree from figure 4(c) is used the segmentation will be different. This segmentation was based on a onset delta tree over the entire piece instead of just the first eight bars

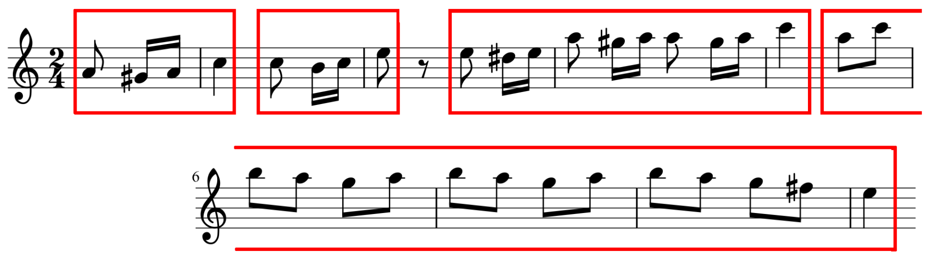


Figure 2: First four segments of piano sonata KV331 III by Mozart (grace notes are not shown)

The pitch interval segmentations are less reliable than the IOI segmentations but since we use pitch only to subdivide IOI segments we can still be sure that some constituent transitions are based on the more reliable IOI segmentation.

4.3 Constituent Features

We can now convert a piece of music into a series of constituents. These constituents will be used to predict expression so we must be able to charaterize them in a way that correlates with the way they are performed. Analogous to YQX we are looking for the *context* of the constituent as well as some description of the constituent itself.

YQX uses a set of three score features: pitch interval, duration ratio and I-R arch. The pitch interval is simply the difference in pitch between the current note and the next note. The duration ratio is the logarithmic ratio of the duration of the current note and the duration of the next note. The I-R arch is is the distance to the nearest point of closure, where closure is calculated from the Implication-Realization analysis [**?**].

We can generalize pitch interval and duration ratio per note to *context features*: **mean pitch interval** and **mean duration ratio**. Definitions can be found below.

Mean pitch interval The difference between the mean pitch of the current constituent and the mean pitch of the next constituent, zero if the current is the last constituent

Mean duration ratio The logarithmic ratio between the mean note duration of the current constituent and the mean note duration of the next constituent

Since I-R arch is related to note-level expression it does not generalize well to a constituent level feature.

The two features above provide information about the constituent context: if they are both zero the constituent is apparently similar in mean pitch and mean duration. At note level there is not much to say about the current note besides the pitch and duration. However at constituent level we would also like to say something about the constituent itself. For this purpose the *constituent features* **mean delta pitch** and **mean delta duration** are used. These features say something about the amount change in pitch and the amount of change in note duration:

Mean delta pitch The mean of all absolute pitch intervals within one constituent.

Mean delta duration The mean of all absolute differences in duration of succeeding notes within one constituent

Note the difference between these features and the context features. The context features use the mean pitch of one constituent and the the mean duration within constituent and compare these to the mean pitch and mean duration of the next constituent. The constituent features use the average of all pitch intervals within one constituent and the average of all differences in duration within one constituent, giving a measure of the spread of pitch and differentation of rhythm within the constituent.

The complete set of score features consists of the two context features and the two constituent features.

4.4 Expression Parameters

Every constituent will be assigned expression parameters that indicate how the constituent is played expressively in a performance. These parameters define what we mean by structure level expression and should therefore be chosen carefully.

Some concepts that we think fall under structure level expression are *crescendo* or *decrescendo*, *ritardando* and *piano* or *forte* etc. Concepts like crescendo, decrescendo and ritardando arguably fall under note level expression and for simplicity we will not consider them structure level expression. In fact, we will only look at the mean tempo, the mean articulation and the mean loudness of a constituent.

YQX defines expression per note in three parameters: *IOI ratio*, *articulation* and *loudness*. These parameters are defined as the logarithmic ratio between the performance IOI and the IOI notated in the score (calculated using some base tempo), the silence after a score note divided by the silence after the performed note and the logarithmic ratio between the loudness of the performed note and the mean loudness of the performance.

We are going to define our own expression parameters in a similar way but let us first look at some issues with the definitions YQX uses. The definition of articulations seems to be awkward and inconsistend. Awkward because if a score note is not followed by a rest, the notated silence after it is zero, rendering articulation undefined. Inconsisted because all the other features use logarithmic ratios and we see no reason not to use a logarithmic ratio for articulation as well.

IOI ratio and loudness are defined relative to mean performance tempo and loudness. However, to capture micro expression in the form of small changes in onset relative to the beat, it seems more logical to define this feature relative to the local tempo, instead of relative to the global tempo. The same argument can be made for defining loudness relative to local loudness instead of global loudness. Structure level expression may help to define the concepts of local tempo and local loudness as we can simply take the mean tempo and loudness within one constituent and take this to be the local tempo and loudness.

The expression parameters that we will use are:

Mean Tempo Ratio The logarithmic ratio between the mean tempo within the constituent and the base tempo of the performance.

Mean Articulation The logarithmic ratio between performance IOI and the score IOI if the next note is not a rest. If the next note is a rest we use the note duration calculated with the score duration of the note and the local expressive tempo instead of the performance IOI. IOI is the onset of the next note minus the onset of the current note.

Mean Loudness Ratio The logarithmic ratio of the mean loudness within the constituent and the base loudness of the performance

Admittedly, we still use mean loudness and tempo to calculate tempo and loudness ratio. This is a symptom of using a segmentation instead of hierarchical structure. When using hierarchical structure, these parameters could be defined relative to the parent constituent.

4.5 Model

We can now reduce a piece of music, represented as a sequence of notes N_{ij}, of which we have a score and a performance, to a sequence of *score feature vectors* F and *expression parameter vectors* E. First, we segment the score into constituents:

$$\texttt{segment}(N_{ij}) = \{c_1, c_2, \cdots, c_n\}$$

We can then extract feature and parameter values from the score and the performance:

$$f_i = (p_i, d_i, \Delta p_i, \Delta d_i)^T$$
$$F = \{f_1, f_2, \cdots, f_n\}$$

where f_i is a feature vector, p_i is the the mean pitch interval, d_i is the mean duration ratio, Δp_i is the mean delta pitch and Δd_i is the mean delta duration. And the expression parameters:

$$e_i = (t_i, a_i, l_i)^T$$
$$E = \{e_1, e_2, \cdots, e_n\}$$

where e_i is a parameter vector, t_i is the mean tempo ratio, a_i is the mean articulation and l_i is the mean loudness ratio.

We can construct a corpus by parsing a number of works in this manner. Given such a corpus, the rendering of a performance can be formulated as maximizing the probability: $P(E|F)$. We can estimate an approximation of this probability by calculating feature likelyhoods and transition probabilities. We approximate the features likely-hood, which is the conditional probability of a feature vector f_i given an expression vector e_i, with the following probability:

$$P(f_i|e_i) = \frac{c(f_i, e_i)}{c(e_i)} \tag{1}$$

Where $c(x)$ is the number of occurences of x in the corpus. The expression transition probability is approximated by the unigram count of e_i divided by the bigram count of e_{i-1}, e_i.

$$P(e_i|e_{i-1}) = \frac{c(e_{i-1}, e_i)}{c(e_{i-1})} \tag{2}$$

where we make the simplification that expression in one constituent is only dependend on the expression in the previous constituent which is of course not true.

The problem of creating a performance of a score is now reduced to finding a suitable sequence of expression vectors given a sequence of feature vectors. We can calculate the probability of one expression vector of a performance as the product the probabilities defined in equations 1 and 2. The probability of the entire performance is approximated by the product of the individual expression vector probabilities.

$$P(E|F) = \prod_{i=1}^{n} P(f_i|e_i)P(e_i|e_{i-1}) \tag{3}$$

The resulting model is analogous to a stochastic part of speech tagger based on a hidden Markov model where we have substituted words for score feature vectors and parts of speech for expression parameter vectors. Rendering a performance is like finding the most likely part of speech tags for a sequence of words. In our case:

$$E^* = \texttt{argmax}_E P(E|F)$$

which can be found using Viterbi decoding.

5 Implementation

This section will discuss practical issues that we faced creating a complete performance rendering system based on the method described in the previous section.

5.1 Corpus and Representation

We were lucky to find and receive permission to use the CrestMusePEDB [3], which is a dataset that contains expressive performances of Western classical music by famous pianists. The music includes works by Bach, Beethoven and Chopin.

Every performance is accompanied by an XML file containing information on how every note from the score is performed. This information consists of a loudness deviation, attack deviation and release deviation. The loudness is defined relative to a base loudness. The attack and release deviation are defined as the portion of a local beat duration that the attack and release deviates from the score.

Local tempo is defined for every beat in every measure as the ratio of the tempo in that beat and the base tempo.

To prepare a score and performance from the corpus for use in our system we use the score to extract melody notes. We do this simply by taking notes in the highest voice in the top staff. From now on when we talk about pieces and notes we mean melodies and melody notes.

Attack and release are clearly note level parameters so we do not use them. We only need tempo deviations and loudness deviations. The average tempo of a constituent is determined by the average tempo deviation of all the beats that fall within the constituent. The average loudness is determined by the average loudness deviation of every note within the constituent.

The deltafunctions that are used in the segmentation process use pitch intervals and duration ratios. The pitch intervals use MIDI note numbers which range from 21 to 108 and go up one semi-tone with each step. Durations are in milliseconds, calculated from the score and a standard tempo, set at 120 beats per minute.

5.2 Discretization

We chose to make the model discrete. We discretize expression and scorefeatures in a different way.

Expression Discretization of expression parameters is a delicate subject, we want to capture small changes in dynamics and tempo precisely, but outliers may fall in large bins. A sigmoid function is very useful for this purpose. The expression parameters are all logarithmic ratios so they theoretically vary from $-\infty$ to ∞. Therefore we first normalize the parameters by dividing through the mizimum absolute value found in the corpus. Discretization of a normalized expression parameter p into d bins is now done as follows:

$$D(p) = \texttt{floor}\left(\frac{d}{1+e^{-sp}}\right) \tag{4}$$

Where $D(p)$ is the discretization of p, s is a special sensitivity parameter indicating how small the changes in tempo that the discretization captures can be; a larger s means a more sensitive discretization. Undiscretization is the reverse operation:

$$D^{-1}(p) = s^{-1}d^{-1}(-\log(p^{-1}) - 1) \tag{5}$$

Features To discretize the features, we simply normalize every feature dividing through the maximum absolute value of that feature found in the corpus. After normalization we multiply the feature by a discretization parameter d that determines the number of bins and take the floor.

$$D(f) = \texttt{floor}(f * d) \tag{6}$$

Where f is the normalized feature value.

5.3 Smoothing

Despite having discretized our feature and observation vectors we often find that we observe feature vectors in a new score that we had never seen during training. We do not want the conditional probabilities from equation 1 to become zero so we have to smooth these probabilities. We use a smoothing technique known as simple Good-Turing smoothing. The idea is that we use the probabilities of things we have seen once for the things we have never seen. Recall how we calculate the conditional probability of a feature vector:

$$P(f_i|e_i) = \frac{c(f_i, e_i)}{c(e_i)}$$

Let us call the $c(f_i, e_i)$ the coincedence count. Let N_c be the number of things with frequency c in the corpus. For example if there are five coincedences (f_x, e_x) nd no other pair of f and e occurs five times, then $N_5 = 20$. Good-Turing reëstimates counts according to this formula:

$$c^* = (c+1)\frac{N_{c+1}}{N_c} \tag{7}$$

The smoothed probability of some event x is

$$P(x) = \frac{c^*}{N}$$

In our case, the probability we seek is $P(f_i|e_i)$, the sample size N is therefore the number of times we have seen e_i: $c(e_i)$. N_c is the number of coincedences with e_i that we have seen c times. The count c is $c(f_i, e_i)$ and c^* is derived using equation 7. The

$$P(f_i|e_i) = \frac{c^*}{c(e_i)} \text{ for all } c > 0$$

By applying this formula to all counts larger than zero we reserve approximately $\frac{N_1}{N}$ probability mass for things that we have never seen. We assign an equal probability for all unseen feature vectors, namely:

$$P(f_{\texttt{unseen}}|e_i) = \frac{1}{U}\frac{N_1}{c(e_i)}$$

Where U is the number of unseen coincedences, which is determined by the number of different observations in the corpus plus the new observations from the score minus the number of coincedences with e_i.

In order to make this work we cannot use N_c directly since it will not be defined for every c. We use a least square apprixmation using the following function.

$$\log(N_c) = a + b\log(c)$$

This completes the definition of simple Good-Turing smoothing. However, even now there may still be some expression parameter vectors that occur with unique feature vectors, so for that expression only N_1 is defined. We cannot fit a function to one sample, so if we have only one Nc sample, we simply turn off Good-Turing smoothing and accept the fact that some probabilities will be zero.

5.4 Performance rendering

Our model is able to generate expressively performed melodies but does not handle polyphony. During training, the bass and harmony notes were stripped off. After rendering an expressive performance we can simply put them back in and estimate their expressive parameters. We do this by this by by giving each bass and harmony note the expressive parameters of the last played melody note.

6 Results

We compiled two corpora from our the CrestMusePEDB. One contained 42 performances of 13 piano works by Mozart, the other contained 49 performances of 19 works by Chopin. For more details on each corpus see appendix A. We suspect that performances of Mozart's music most outspokenly use structural level expression. Expression within a constituent tends to be more unsteady in performances of Chopin's music.

To test the system we trained it on one of the two corpora and let it perform a number of works. Here we show the results of four renderings of Chopin's Mazurka No. 19, Op. 30-2 and Mozart's Piano Sonata 310 III. To see what the effect was of each corpus on different styles of music we performed each work on both corpora. For reference, we also show a performance by one of the pianists in the corpus. Given the small dataset and the free nature of expression we did not expect the performances to correlate much with performances by human pianists.

For all four performances the system was trained with a discretization factor of five. In practice, this means that the constituent features (see section 4.3) were discretized into five bins and the context features into a slightly larger number of bins since their values can be negative as well. The expression features where discretized into ten bins, with a sensitivity of five. The bins had to be this large to recognise most of the score features in the score that was to be performed.

Segmentation was done using only delta onset trees. This proved to be most reliable for Mozart.

When rendering a Mozart performance on the Mozart corpus and a Chopin performance on the Chopin corpus any of the performances of the piece that had to be performed were left out of the corpus during training.

Plots showing dynamics, tempo and articulation of the resulting performances are shown in figure 3 and 4. The values on the y-axis, labeled deviation, are the exponents of the expression parameters, which themselfs are logarithmic ratios. In other words, the values indicated by the axes are ratios. A tempo of 2.0 means twice as fast as the average tempo, an articulation of 0.5 means the notes are played twice as short as notated in the score, a dynamic value of 0.5 means the notes are played half as loud as the average loudness of the performances. Likewise, to translate the output of the system into a performance, the mean tempo and loudness have to be set manually.

The performances are plotted per note. The block like character of the axes is caused by the segmentation. For the sake of comparison, the performances by human pianists have been averaged over each constituent and discretized, so these plots show how the performance is represented in the corpus.

Of course staring at plots of music performances is not nearly as interesting as actually listening to them. For this purpose, a few demo performances including the ones shown here can be found online at [DEMO URL].

7 Discussion

Despite the large bin size, many combinations of features and expression parameters were never encountered during training. The same applies to transitions of expression, only small portion of all possible transitions of expression are actually encountered in the corpus. Some of the feature likelyhoods are smoothed but the transition probabilities are not smoothed at all. Since our model does not allow zero probability transitions, our model is heavily overfitting. The resulting performances only contain changes of expression that were actually encountered in the corpus.

In most applications of machine learning, this degree of overfitting would not be acceptable. However, in this particular application of machine learning we seem to be able to get away with it. Playing a phrase transitions exactly as encountered in the corpus seems to be acceptable as long as we play the right transitions at the right moment. The overfitting and the large bins size give the performances a bit of a stereotypical and cartoonistic feel, but that is mostly caused by the small corpus. Whether it makes musical sense to perform Chopin on a system trained on Mozart is left for the reader to decide.

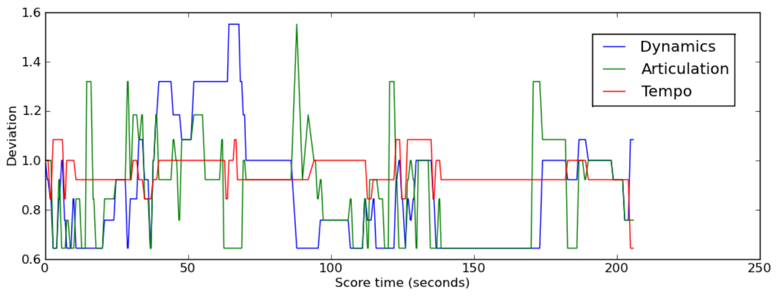
Although the output of the system is polyphonous, the system at its core only uses a single voice for its performances. We use the highest notes in the top staff for this and hope that these are the melody notes. Apart from the fact that this is not always the case, the other notes generally should note be played with the same expression. Bass notes for example are often played slightly softer and more legato than melody notes.

The system does not make use of expressive markings in the score. A mature performance rendering system should incorporate these in some way. Even a human performer is not expected to read the mind of the composer and uses the expressive markings as a guide for performance. Disregarding articulation marks and dynamic markings has a significant impact on the results: the system has an overall slight preference for playing staccato and softly. This is due to the fact that staccato notes or notes that should be played soft are treated as normal notes and bias the statistics.

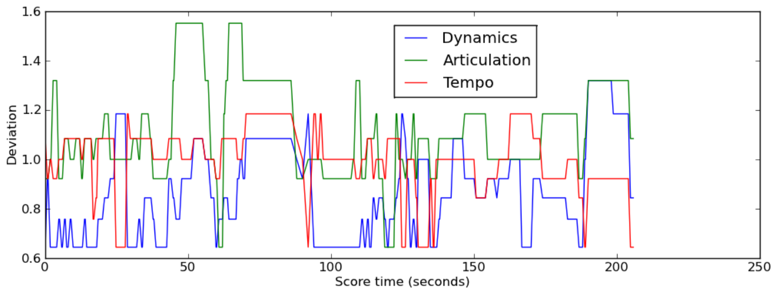
Ignoring note level expression is quite an extreme measure to take when it has been an adequate way to demonstrate the phenomenon of structure level expression. Yet, we cannot deny the need for note level expression in a performance rendering system that intends to completely simulate human expression. Section 9 suggests a possible integration of structure and note level performance rendering systems.

8 Conclusion

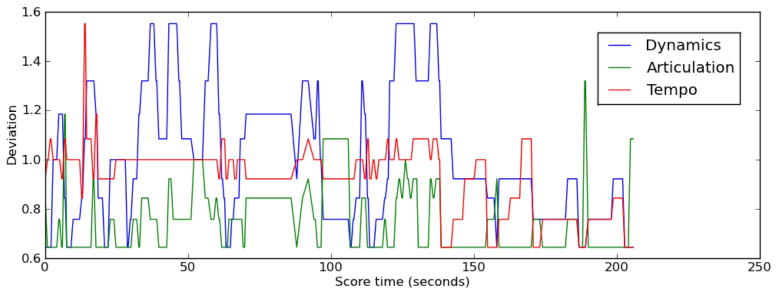
In this thesis we have introduced a way of recognizing and using structure level expression. We argued why we think it is a crucial aspect of performance rendering systems and we critisized note level performance rendering for not being



(a) Performance by the system trained on the Mozart corpus

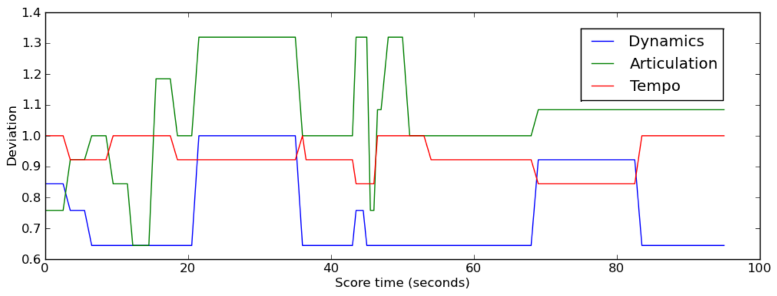


(b) Performance by the system trained on the Chopin corpus

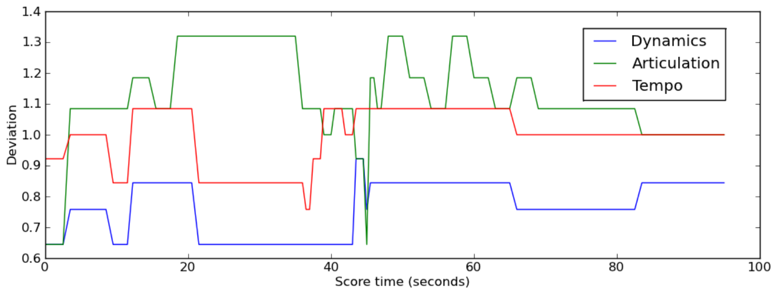


(c) Performance by Maria João Pires

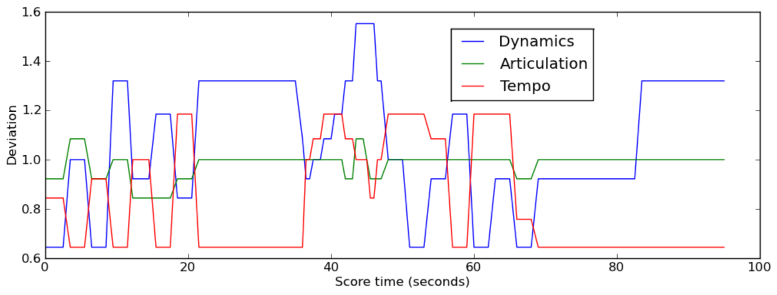
Figure 3: Three performances of Mozart's Piano Sonata 310 III.



(a) Performance by the system trained on the Mozart corpus



(b) Performance by the system trained on the Chopin corpus



(c) Performance by Samson Francois

Figure 4: Three performances of Chopin's Mazurka No. 19, Op. 30-2

able to correctly learn when to make daring and large changes in expression as well as lacking the ability to convey structure level expression.

Our goal was to create a system that was the complement of a note level performance rendering system. A system that dared to make sudden big changes in expression and clearly accentuates musical structure. Our results show that our system indeed is not shy this kind of big expressive changes. The performances of the system contain sometimes smooth, sometimes sudden changes of expression. It is up to the listener to judge whether these changes are actually an improvement over the smoother performances of note level performance rendering systems.

The system that we proposed reduces the idea of structure to a segmentation of the score into constituents based on a structural analysis. For the structural analysis we used the delta framework [7], proposed by Markwin van den Berg. We noticed that a segmentation based on onset delta trees works well for some music but does not recognize all constituent transitions. To overcome this we suggested to use a second order pitch delta tree, but we ended up discarding this idea because it still produced awkward segmentations. Our system would therefore not work very well on music that has a consistend rhythmic structure like some of Bach's music. In the end, an onset tree segmentation worked good enough to produce musically relevant segmentations given the the right genre of music.

To characterize constituents and their relations to other constituents we used constituent likely-hoods and bigram transition probabilities of expression. The resulting hidden Markov model could be used in combination with Viterbi decoding to generate performances. This approach is very similar to stochastic part of speech tagging using hidden Markov models.

9 Future Work

The abilities of performance rendering systems still are not even close to the abilities of real human pianists. In relation to this system we can clearly identify a number of things that would bring the system a little bit closer to the abilities of a human pianist.

The segmentation is not always perfect. The delta framework offers us more analytic power than we have used in this system but we were not able to sensibly combine its analyses into one segmentation apporach. We are unsure how we should combine different delta trees into some sort of consensus.

Such a consensus tree would be especially useful in an extension of our system to a true structure based performance rendering system, namely one that utilizes multilevel structure.

Another open problem is how we should deal with multiple voices. Some proposals have been made in this field. Tae Hun Kim et al. [5] split a piece into melody, bass and harmony and use separate probabilistic models for each of them. When using different expression for different voices, another problem presents itself, namely how to synchronize the voices so that the performance still sounds like it is played by one artist or a group of artists with an adequate notion of timing. Pop-e [4] uses an automatic analysis to determine the attentive part: the voice of the music perceived as melody at any point in the piece. It uses this information to synchronize expression in different voices.

As we mentioned in section 7, our system ignores expressive markings. The result is that depending on the corpus, the performance renderer has unwanted biases. A possible approach to incorporating expressive markings in the system is to determine what, given an expressive mark in the, is the average way to play a given note and normalize this out of the corpus. Say staccato notes are played, on average, half the length of the notated length. We specify articulation deviation relative to this half duration instead of the full duration.

We hope that in the future, as the CrestMusePEDB project develops, data sparsity will be less of a problem. For now however, we are left with having to do discretization and smoothing. The smoothing technique we used, Good-Turing smoothing, is arguably not the best way to smooth the corpus. Good-Turing smoothing assigns equal probabilities to all unseen score features. It is likely that unseen scorefeatures that are very similar to scorefeatures that we have seen should also be played in a similar way as the scorefeatures they resemble. A different smoothing technique may be able to use this information. We could use a smoothing technique similar to Katz-Backoff REF: upon encountering an unseen feature vector, discretize again the corpus into increasingly larger bins until we do find the featurevector in our corpus.

The system would certainly benefit from a notion of repetition and similarity. Repetition is a very good indicator of constituent breaks. Repetition and similarity could also be used to improve expressiveness of performances. It is probably telling when a phrase is repeated three times and then slightly altered the fourth time. Although finding similarity and musically significant repetition is a subject of its own the delta framework could help to define repetition arbitrarily of transposition or rhythm. A list of pitch deltas can for example be used to detect repetition independend of transposition and a list of duration deltas can be used to detect repetition of rhythm independend of the notes used.

The YQX system defines expressive tempo implicitly by predicting the logarithmic ratio of the IOI in a performance and the IOI in a score. Timing alterations of notes are always defined relative to the base tempo. This does not correspond to the intuition that the *the tempo itself* is altered during the performance and that rhythmic changes should be seen relative to the local tempo. The same applies to the way YQX looks at loudness. This is specified as the logarithmic ratio between the notes loudness and the mean loudness of the performance.

An integration of constituent level expression and note level expression can provide a solution to this problem. We can define expressive tempo relative and dynamics relative to the expression parameters of the constituent. A possible approach is to first define constituent level expression and use a note level expression system on each constituent seperately.

Another approach is to use hierarchical structure. Expression of every constituent can be defined relative to the parent constituent. If we let the hierarchy extend to the note level (the delta framework does this) we get a smooth integration of note level performance rendering and structure level performance rendering. In such a system notes are terminal notes in a structure tree, note level expression is specified relative to the parent constituent's expression which is again relative to the its own parent constituent. Ultimately the root node's expression contains the base expressive parameters.

Finally, as we mentioned in section 3 the ability of the system to characterize constituents and their relations to other constituents in a musically relevant way is an important factor influencing the results. We do not have the illusion that a set of four scorefeatures and bigram transition probabilities are a valid model of human musical intuition, let alone producing highly expressive performances. A better notion of repetition, a notion of key, harmony and general musical knowledge are the least that seems to be required for that purpose.

10 Acknowledgements

References

[1] S. Flossmann, M. Grachten, and G. Widmer. Expressive performance rendering: Introducing performance context. In *Proceedings of the SMC 20096th Sound and Music Computing Conference*, pages 155–160, 2009.

[2] A. Friberg, V. Colombo, L. Frydén, and J. Sundberg. Generating musical performances with director musices. *Computer Music Journal*, 24(3):23–29, 2000.

[3] M. Hashida, T. Matsui, and H. Katayose. A new music database describing deviation information of performance expressions. In *Proc. of the 9th International Conference on Music Information Retrieval (ISMIR). Philadelphia*, 2008.

[4] M. Hashida, N. Nagata, and H. Katayose. Pop-e: A performance rendering system for the ensemble music that considered group expression. In *Proceedings of 9th International Conference on Music Perception and Cognition*, pages 526–534, 2006.

[5] T.H. Kim, S. Fukayama, T. Nishimoto, and S. Sagayama. Performance rendering for polyphonic piano music with a combination of probabilistic models for melody and harmony. 2010.

[6] F. Lerdahl, R. Jackendoff, and R.S. Jackendoff. *A generative theory of tonal music.* The MIT Press, 1996.

[7] M.J. van den Berg. Aspects of a formal theory of music cognition. 1996.

A Corpus

Work	Performer
Sonata KV331 I.	Hiroko Nakamura
	Glenn Gould
	Christoph Eschenbach
	Ingrid Haebler
	Lili Kraus
	Maria João Pires
	Alicia De Larrocha
	kn?
	yi?
	tn?
	Norio Shimizu
	mo
	mk?
	ea?
	nm?
	kt?
	tm?
Sonata KV331 II.	Hiroko Nakamura
	Maria João Pires
	Alicia De Larrocha
	ea?
Sonata KV331 III.	Hiroko Nakamura
	Maria João Pires
	Christoph Eschenbach
Sonata KV545 I.	Maria João Pires
	Glenn Gould
Sonata KV545 II.	Maria João Pires
	Glenn Gould
Sonata KV545 III.	Maria João Pires
	Glenn Gould
Sonata KV279 I.	Glenn Gould
	Maria João Pires
Sonata KV279 II.	Glenn Gould
	Maria João Pires
Sonata KV279 III.	Glenn Gould
	Maria João Pires
Sonata KV310 I.	Maria João Pires
	Glenn Gould
	Hiroko Nakamura
Sonata KV310 II.	Maria João Pires
Sonata KV310 III.	Maria João Pires
Sonata KV570 III.	nm?