International Journal on Artificial Intelligence Tools

EXPLORING PIANIST PERFORMANCE STYLES WITH EVOLUTIONARY STRING MATCHING

SØREN TJAGVAD MADSEN
Austrian Research Institute for Artificial Intelligence
Freyung 6/6, A-1010 Vienna, Austria
soren.madsen@ofai.at

GERHARD WIDMER
Department of Computational Perception, Johannes Kepler University
Altenberger Straße 69, A-4040 Linz, Austria
gerhard.widmer@jku.at

We propose novel machine learning methods for exploring the domain of music performance praxis. Based on simple measurements of timing and intensity in 12 recordings of a Schubert piano piece, short performance sequences are fed into a SOM algorithm in order to calculate 'performance archetypes'. The archetypes are labeled with letters and approximate string matching done by an evolutionary algorithm is applied to find similarities in the performances represented by these letters. We present a way of measuring each pianist's habit of playing similar phrases in similar ways and propose a ranking of the performers based on that. Finally, an experiment revealing common expression patterns is briefly described.

Keywords: Self Organizing Map, Evolutionary Algorithm, Approximate String Matching, Expressive Music Performance

1. Introduction

Expressive music performance as the artistic act of 'shaping' a given piece of written music has become a topic of central interest in the fields of musicology and music psychology[1]. In classical music, in particular, the performing artist is an indispensable part of the system, shaping the music in creative ways by continually varying parameters like tempo, timing, dynamics (loudness), or articulation, in order to express his/her personal understanding of the music. Musicologists and psychologists alike would like to understand the principles of this behavior – how much of it is determined by the music, whether there are unwritten rules governing expressive performance, etc. Recently, also AI researchers have started to look into this phenomenon and to apply their techniques (e.g., machine learning) to get new insights into patterns and regularities in expressive performances[2,3].

In this paper, we present an evolutionary algorithm for finding approximately matching substrings in character sequences, and use it to search for structure in expressive performances (by famous pianists) encoded as strings. The goal is to

study both the artists' *intra-piece consistency* and potential *similarities* between their playing styles.

It is known (and has been shown in laboratory experiments) that performing expressively in a stable manner is a way of emphasizing the structure of the music[4]. In particular, similarities in timing patterns across repeats have been noted in virtually every study in the field[5]. While the above studies were mainly based on measurements of time alone, we also expect this type of behavior (similar types of phrases being played with distinctive recognizable performance patterns) when doing a joint examination of timing and dynamics in music performance. One goal of our experiments is to compare 12 famous pianists according to the extent of stability in their performance – their *intra consistency*. This can be understood as the extent to which it is possible to distinguish musically similar phrases based on their interpretation alone. We propose a measure of this phenomenon and rank the pianists accordingly. A second goal is to compare pianists' performances directly, revealing examples of commonalities in the performance praxis.

One approach to attack these problems is to perform a close examination of the performances of designated repeated patterns (the approach taken, e.g., by Repp[5] or Goebl et. al.[6]). We do our investigation in the reverse order – finding the sequences of greatest similarities in the performances and comparing the music behind. This approach takes its starting point in the performance rather than the music. In this way, we expect the investigation to be less biased by a predetermined way of perceiving the music.

2. Performance Data Acquisition and Representation

The data used in this experiment comprises recordings of Franz Schubert's *Impromptu*, D.899 no. 3 in G♭ by 12 different pianists (see Table 1). The performers' different choices of tempo are immediately apparent from the difference in duration of the tracks. The slowest interpretation by Kempff lasts 6:47 min., and the fastest by Lipatti 4:51 min.

We want to characterize the performances in terms of measurements of two parameters: tempo and loudness. We do this by extracting these features as discrete measurements for each beat in the score. The 12 recordings were semi-automatically 'beat-tracked' with the aid of appropriate software[7,8]. The onset time of each beat was registered and for each beat a local tempo in beats per minute was computed. Furthermore the dynamic level at each beat tracked was also computed from the audio signal[9].

For each beat in the score, we now have measurements of tempo and loudness, forming a bivariate time series. The performances can be described as consecutive points in the two dimensional tempo-loudness space as suggested by Langner & Goebl[10]. A graphical animation tool called *The Performance Worm*[11] displays such performance trajectories in synchrony with the music. A part of a performance as visualized by the Worm is shown in Figure 1. A movement to the right signifies

Table 1. The recordings used in the experiment.

Index	Pianist	Recording	Year
0	Barenboim	DGG 415 849-2	1977
1	Brendel	Philips Classics, 456 061 2	1972
2	Gulda	Paradise Productions 9/99	1999
3	Horowitz	Columbia MS 6411	1962
4	Kempff	DGG 459 412-2	1965
5	Leonskaja	Teldec 4509-98438-2	1995/96
6	Lipatti	Emi Classics CDH 5 65166 2	1950
7	Maisenberg	Wiener Konzerthaus KHG/01/01	1995
8	Pires	DGG 457 551-2	1996
9	Rubinstein	BMG 09026 63054-2	1991
10	Uchida	Philips 456 245-2	1996
11	Zimerman	DGG 423 612-2	1990

an increase in tempo, a crescendo causes the trajectory to move upward, and so on. Note that the display is interpolated and smoothed. For our experiments only the actual measured points were used. The discrete version of the performance captures only the fundamental motions in the tempo-loudness space but hopefully also the fundamental content of the performances. Accentuations between the points of measurements are not present in the data. Neither are refinements of the expression such as articulation and pedaling.

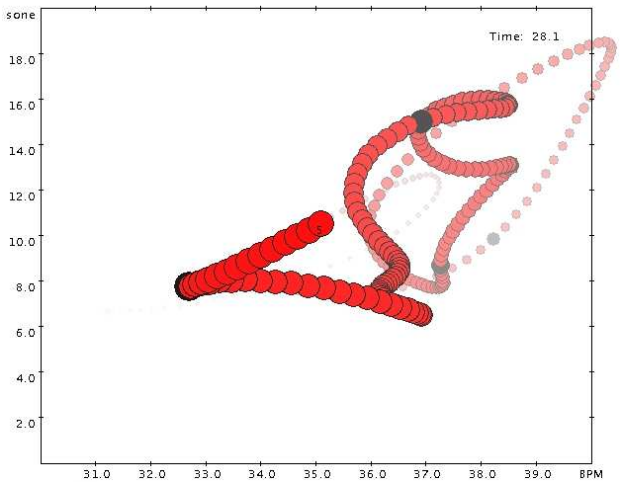

Fig. 1. Snapshot of the *Performance Worm* at work: First four bars of Daniel Barenboim's performance of Mozart's F major sonata K.332, 2nd movement. Horizontal axis: tempo in beats per minute (bpm); vertical axis: loudness in *sone*. Movement to the upper right indicates a speeding up (*accelerando*) and loudness increase (*crescendo*) etc. etc. The darkest point represents the current instant, while instants further in the past appear fainter.

2.1. *Performance letters*

The idea is now that performance styles can be compared by comparing sequences in the points measured. This task has already been attacked with machine learning methods working on features extracted from subsequences.[12]

Instead of analyzing the raw tempo-loudness trajectories directly, we chose to transform the data representation and thus the entire discovery problem into a form that is accessible to common inductive machine learning and data mining algorithms: string analysis. To that end, the performance trajectories are cut into short segments of fixed length. The resulting segments are then grouped into classes of similar patterns using clustering. The clustering is generated by a self-organizing map (SOM) algorithm. A SOM generates a geometric layout of the clusters on a two-dimensional grid or map, attempting to place similar clusters close to each other. For each of the resulting clusters, a prototype is computed. These prototypes represent a set of typical elementary tempo-loudness patterns that can be used to approximately reconstruct a 'full' trajectory (a complete performance). In this sense, they can be seen as a simple alphabet of performance, restricted to tempo and dynamics.

Figure 2 displays a set of prototypical patterns computed from the 12 performances of the piano piece. Based on some experiments the map layout was set to be a 5×5 grid, and the size of the input segments set to represent a duration of two beats. Each performance was thus divided into segments of three points each – the end point of a sequence being the beginning of the next. The duration between the first and the last of three measured points is equal to two inter-beat intervals, which is normally referred to as two 'beats'. The figure displays the resulting 25 prototypes after clustering all segments from all performances with the SOM algorithm. The prototypes are labeled with letters – the *performance letters*. Each performance can now be represented as a string of 170 letters. One such performance is shown in Figure 3.

Deciding the number of clusters as well as the length of the input segments is decision we made based on a few experiments. Changing these values may increase or decrease the descriptive power of the letters. A more thorough investigation of these dependencies remains to be done, in order to examine the possibilities of improving the results we describe here.

Another choice we had to make is what type of normalization we should apply to the raw measured data prior to clustering. Due to the differences in recording quality or producers' ideals, the dynamic average and range of the performances may differ quite a lot. To prevent this recording artifact from having influence on our data, we decided to divide every dynamics value by the global mean of all dynamics values measured. Likewise the pianists' choice of tempo may be quite different, so in order to obtain globally comparable conditions some normalization should be applied to the tempo values as well. With no normalization one could expect the clustering algorithm to cluster e.g. the sequences from the fastest performance into a cluster

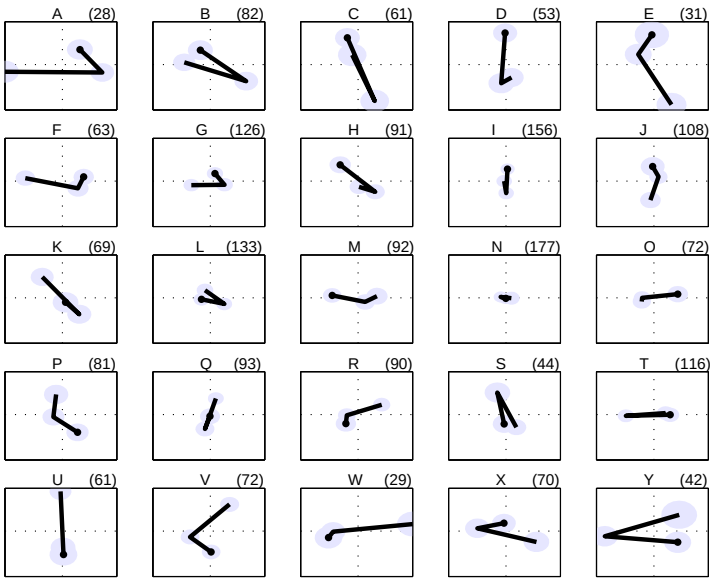

Fig. 2. The clustered performance segments. Each cluster is labeled with a *performance letter*; the number of occurrences of each letter is written in parenthesis. The shaded regions indicate the variance within a cluster

```
GTINIMPTLNHNQNLTINQJHNUQLOINMNHSHFQQLEVPROLNMNLECCPPSEKCPQNYNTLTTINRL
ODPRODPIECCHUJPOECCIUJVRTGXTGOQITIDQDRYFTGNGJPTLOINNNLIHOIILJUQLINTMT
JNVGLOXTIDIEUURGIDIDUVRGNHMOINLW
```

Fig. 3. Daniel Barenboim playing the Impromptu.

on its own. However we want to be able to characterize all performances with the same set of descriptors, so likewise all measured tempo values were divided by the global mean.

The prototypes are a rough approximation of the group of subsegments that they represent. Note the letter `N` in the alphabet in Figure 2. It is the most frequent letter overall. Since all three points are very close together, it describes many situations where almost no changes in dynamics nor tempo occur, and hence probably has the least descriptive power. However as we shall see the letters all together do contain valuable information. It has even been shown that it is possible to recognize performers on the basis of these letters.[13]

Finding similarities in the performances can now be expressed as a string matching problem or an approximate string matching problem, which is the main subject of this paper. The approximate string matching is done with an evolutionary algorithm described below.

In other words we want to discover patterns in the performance string. A pattern

is a substring that occurs at least twice (identically or with some variation). We want to discover these patterns as well as find all occurrences of them. We will refer to the task of finding all similar non-overlapping strings in a performance (up to a similarity threshold) as *segmenting* the string, yielding a *segmentation* of the performance.

3. Measuring Consistency

Before introducing the search algorithms, we need to explain how we evaluate a segmentation. Much music is intended to be heard phrase by phrase. Since music unfolds in time, we are generally not capable of keeping focus on all we have heard since the beginning of the piece. Rather the music is composed in such a way that we listen to the music one phrase at a time. A phrase unfolds in a shorter time period and is structured by the illusion of boundaries created by logical musical entities in the musical content. The Schubert *Impromptu* is like this. Furthermore a music piece typically contains a high degree of repeated material. In fact structure of a piece can be derived when decomposing the composition into smaller sections and subsections or *phrases*.

Given a segmentation of a performance we are interested in examining to what extent pianists have played similar sounding phrases in the piece with similar expressions. We will call this phenomenon *consistency*. The word is not meant to imply an assessment of value. We would like to use it simply as a measure of the degree to which similar phrases are played with similar interpretations, and dissimilar phrases given different interpretations. It does not tell anything about the pleasantness or quality of the performance per se, although some listeners might find these related (related to consistency *or* inconsistency).

Measuring consistency is done in two steps: First the performance string is examined for recurring patterns (similar or approximately similar). This is done by string matching algorithms described below. A number of different patterns may be found – each pattern having two or more instances (occurrences). Next we evaluate how well the discovered similarities in the expression correspond to similar sounding phrases in the music. A performance will result in a 'perfect' segmentation and therefore a perfect evaluation, when a performer distinguishes every type of phrase with it's own expression.

Given two similar sequences of letters we would like to be able to count how many letter positions (each representing half a bar of music) the sequences have in common that refer to similar music. For that a structural analysis of the piece was performed (by the authors, see Table 2, dividing the piece into similar sections and subsections (phrases) based on the musical content alone.

The *Impromptu* can be considered to consist of 15 different phrases of varying length (1 to 4 1/2 measures), some of which occur up to six times (with some variation). The phrases are labeled with letters a to l. For most types of phrases, all instances have the same length – e.g. all 4 occurrences of phrase a have length 4.

Table 2. The structural analysis of the *Impromptu*

Position number	0	4	8	12	16	20	24	29	32	36	40	45	48	52	54	58	60	62	68	72	77	85	94	100	104
Measure number	1	3	5	7	9	11	13	15	17	19	21	23	25	27	28	30	31	32	35	37	39	43	48	51	53
Form section	A				B				B				C					D			E		D'		
Subsection	a	b	a	c	d	d	e	f	d	d	e	f	g	h	g	h	h	i	j	j	k	k	i	j	l
Duration (positions)	4	4	4	4	4	4	5	3	4	4	5	3	4	2	4	2	2	6	4	5	8	9	6	4	4

Position number	108	112	116	120	124	128	132	137	140	145	146	150	152	154	158	160	162	164	166	168
Measure number	55	57	59	61	63	65	67	69	71	73	74	76	77	78	80	81	82	83	84	85
Form section	A				B'						C									
Subsection	a	b	a	c	d	d	e	f	e	f	j	m	n	j	m	n	n	n	o	o
Duration (positions)	4	4	4	4	4	4	5	3	5	1	4	2	2	4	2	2	2	2	2	4

But occurrences of phrases j, k and f have differing lengths. In these cases the analysis is made in such a way that all phrases begin in the same way – the longer being continuations of the shorter.

Using the table we can ask if two letter positions refer to similar music. They do if they are positioned in the same type of subsection and with the same offset from the beginning in that subsection. Consequently we can count how many letters two performance substrings have in common that refer to similar music. We are going to use this for measuring the overall consistency of a segmentation.

The fact that one section (l) is never repeated together with the fact that similar phrases can have different lengths means that a 'perfect' segmentation can maximally have 163 of the total 170 positions matched correctly.

A segmented performance string is a sequence of letters and patterns (substrings). Here is an example of a segmented string – the repeated pattern labeled '1' has been identified:

A B [1: C D E F] G [1: C D E F] H I J

In general, let a segmentation return m different patterns with n_j occurrences of each pattern: $s_1^1 \ldots s_{n_1}^1$, $s_1^2 \ldots s_{n_2}^2$, $\ldots$, $s_1^m \ldots s_{n_m}^m$. Every string s_j^i sits somewhere in the performance string. The search algorithms we have used allow the patterns to be nested (an instance of pattern i may be contained in an instance of pattern j, $i < j$). This is because the string matching algorithms we use actually discover hierarchies in the input strings. A segmentation of a string is a nested structure, constructed bottom-up.

The evaluation scans through the segmentation from left to right. Every letter which is not matched to any other is counted as unmatched. When a pattern is met, it is compared for consistency with the other instances of the same pattern one at a time. We calculate the maximal match among these:

$$maxMatch(s_{n_j}^i) = \max_{s_{n_k \neq n_j}^i} countMatch(s_{n_j}^i, s_{n_k}^i) \tag{1}$$

where the countMatch method is scanning through the string positions pairwise, counting how many positions correspond to similar music according to the analysis.

If a string s_j^i of length p was found to correspond in q positions with another string s_k^i (from the same class i) ($q \leq p$) it results in a score of q true positives (TP) and $q - p$ false positives (FP).

When scanning through the top level of the segmentation we get exactly one evaluation of each position so that invariably $TP + FP + unmatched = 170$.

Given different segmentations, we can now measure how well they correspond to the structure of the music. We express this in terms of *recall* and *precision* values[14]. Recall is the number of correctly found letters (TP) divided by the total number of letters there is to be found in an 'optimal' segmentation (in this case 163). Precision is TP divided by the total number of matching letters found (TP + FP). The F-measure combines recall and precision into one value (an α of 0.5 is used throughout this paper giving equal weight to precision and recall):

$$F(R, P) = 1 - \frac{1}{\alpha \frac{1}{P} + (1 - \alpha)\frac{1}{R}} \ , \ \ 0 \leq \alpha \leq 1 \tag{2}$$

The F-measure weights the number of correct corresponding letters found against their 'correctness'. As precision and recall improve, the F-measure reflecting the *in*consistency drops (the consistency increases).

4. String Matching

To inspect the performances for distinguishable patterns, we now are interested in finding recurring substrings in the performances.

4.1. *Exact string matching*

A natural way to start this task was to apply an exact string matching algorithm. The SEQUITUR algorithm[15] that identifies hierarchical structure in sequences was applied to each string.

Distinctive similarities in the performances do not show right away. The algorithm found mainly very short sequences and many letters were not included in any sequence. Even though the longest repeated patterns found in all of the performances spanned 5 letters (2 1/2 measures of music), in some of the performances only repeated strings of 2 and 3 letters were found. Figure 4 shows 2 occurrences of a 5 letter pattern found, plotted in the tempo-loudness space as well as tempo and loudness separately. The performances appear less similar in the tempo-loudness space due to the accumulated inaccuracies from the two dimensions.

These two sequences do refer to similar phrases in the music. Most of the strings found similar were however not referring to the same music. With no exception, in every segmentation the number of true positives was smaller than the number of false positives (precision below 0.5). The segmentations of the performances by Lipatti and Rubinstein were found most precise (45.5 % and 43.5 % respectively). Also the greatest recall rates were found in these performances, which therefore score the best (lowest) F-measures (0.652 and 0.686).

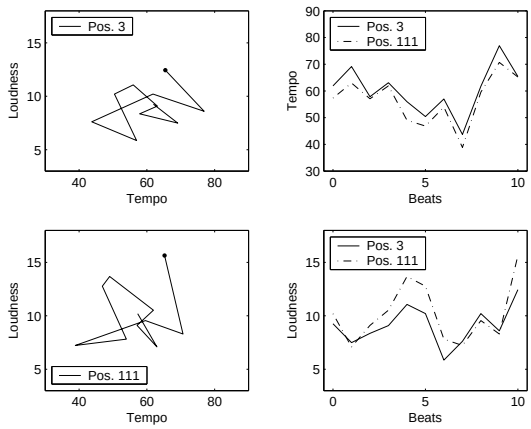

Fig. 4. Two instances of the letter sequence 'LHPTB' from Rubinstein's performance plotted in the tempo-loudness space (left) and each dimension separately (right).

From this first attempt it looks as if the pianists are playing rather inconsistently, only occasionally repeating a short performance pattern. Segmenting the performances based on exact matching might be expecting too much consistency of the performer and indeed expecting too much of the discrete approximate representation of the performance. On the other hand, longer strings do occur so the performance letters seem to be able to represent some characteristics in the performances. We will now explore the possibilities of finding similar patterns based on inexact string matching.

4.2. *Approximate string matching via evolutionary search*

Searching for approximate strings in the performance data resolves to locating letter sequences that are as similar as possible. We are only interested in finding strings of the same length, which makes the comparison method straight forward. Comparing two strings is done by comparing their letters pairwise. Using the letter distance matrix output from the clustering process, we get a more nuanced picture of the similatities of the strings than a simple hamming distance.

Having the comparison procedure, the next thing is to locate similar strings. Since we do not know which words we are searching for, an exhaustive search would require comparing all possibilities of non overlapping strings of the same length to each other.

To reduce the computational complexity we took a stochastic approach. We have developed an evolutionary search algorithm able to find approximately matching strings. In our first experiments we want to find the longest most simlar strings in performances. These objectives can easily be expressed in the fitness function of the EA. We can likewise add more constraints to the search objective (see chapter 5.3).

The algorithm operates on the input string represented as a double linked list of

objects. There are two types of objects in the list: objects containing single letters (unmatched letters) and compound objects each representing an instance of an already found pattern (substring). Before segmenting, all objects are single letters.

An individual of the population of the EA is a 'guess' that two subsequences in the list of equal length are similar. A subsequence is simply a subpart of the input list. A subsequence represents a string which can be constructed by concatenating the letters and strings represented in the list of objects in such a subsequence. An individual thus points at two *subsequences* representing two *substrings* – we will use both terms in the following.

Each individual points at any time at two such subsequences, and the sequences are required to have the same size (number of objects). However two different individuals need not point at equal sized subsequences.

The fitness function decides which individuals are the most successful guesses based on string similarity and size of the subsequences. The best guesses are selected for the next generation and by doing crossover and mutation in terms of altering the guesses (dynamically changing the size and position of the subsequences) the algorithm can enhance the fitness of the population. After some generations the algorithm hopefully settles on the globally fittest pair of subsequences in the search space.

The EA selects 50% of the new generation by tournament selection from the old generation (tournament size of 2). In addition the overall fittest individual always survives (elitism). 10% are made by crossover of the selected individuals, and the last 40% are found by mutating the selected and crossbred individuals. Finally random mutation is also applied to 40% of the new population.

The crossover of two individuals selects at random one sequence from each parent and makes a new individual with these sequences. Since the sequences may have different sizes (coming from different individuals), the longest is shortened until the strings have equal size.

Mutation of one individual consists of applying one of four different operators. All operators preserve the invariant that both subsequences in each individual have the same length.

- Substitute: Substitute one sequence in the individual with a randomly chosen new sequence of same size. Probability $p = 0.05$.
- Extension: extend both sequences one object either at the front or at the end, chosen at random and independently for each sequence. Probability $(1 - p)/3$.
- Shrink: remove an object at the beginning or end, chosen at random and independently for each sequence. Probability $(1 - p)/3$.
- Slide: move the start pointer of the sequence one position to the left or right. Probability $(1 - p)/3$.

The fitness function has a dual goal: to optimize the string similarity and to prefer longer strings to shorter ones. This is expressed as a minimization problem. The

fitness calculation performs a pairwise letter to letter comparison of the letters in the strings and sums up the distances based on the distance matrix output in the clustering process. This is the basic string similarity measure. The string size contributes to the fitness in such a way that longer strings are valued higher than shorter ones. This is to bias the algorithm towards considering longer less similar strings to short exact ones. The preference for size is implicitly implemented as an *average dissimilarity allowed* (δ) per letter in the strings. Actually a third goal of the fitness calculation is to prohibit overlap between subsequences in an individual – completely overlapping subsequences are maximally similar, but not interesting, so overlapping result in an unfeasible evaluation.

Segmenting a performance now consists in iteratively pattern discovery in the performance string. In each iteration we run the EA and obtain a fittest pair of strings and their fitness value. A threshold determines if the fitness value is low enough for the strings to be claimed similar and be part of the segmentation. In that case, a search for more occurrences of each of the strings is executed. When no more occurrences can be found, every subsequence representing a discovered string is substituted in the data structure with compound object now representing the pattern. The compound objects are given a number (the iteration number) identifying this class of performance pattern. Further searches in the data can include and expand these already found entities – nesting can occur. Compounds with the same identifier are regarded as having zero distance to each other. The discovering of new patterns continues until the most fit pair of strings found are not within the threshold.

We can now express the fitness calculation: Let a and b be the subsequences to compare, and a_i and b_i represent the i'th object (letter or compound) in the subsequence:

$$sim(a,b) = \sum_{i=1}^{|a|} dist(a_i, b_i)\,, \quad |a| = |b| \tag{3}$$

$$dist(x,y) = \begin{cases} matrix(x,y) - \delta & \text{if } x \text{ and } y \text{ are both letters,} \\ 0 & \text{if } x \text{ and } y \text{ are compounds of same type,} \\ P & \text{if } x \text{ and } y \text{ are compounds of diff. type,} \\ P & \text{if } x \text{ or } y \text{ is a compound and the other not.} \end{cases}$$

The value P is a penalty value, which should be set high (resulting in an evaluation above the threshold) in order to keep the algorithm from allowing different patterns to be similar and letters to be equal to substrings.

How much difference we want to accept between the strings is controlled by the δ value and the threshold. Setting the parameters too conservatively (e.g. setting δ and threshold to zero), leaving no room for near matches, would make the algorithm

behave as an exact matching algorithm. On the other hand, allowing too much difference would make the algorithm accept anything as similar.

In the experiments described here, we have fixed the threshold to zero, so strings a and b are considered similar if $sim(a, b) \leq 0$. The dissimilarity allowed then depends solely on the δ value. The normalized letter distance matrix output in the clustering process contains values in the interval [0;1]. Generally there is a distance of 0.17–0.39 between letters next to each other on the normalized 5×5 SOM. The δ value should also be given a value in [0;1]. We do systematic experiments with different δ values in the next chapter.

We saw above that segmenting according to exact matches was apt to point out numerous small sequences, which tended not to reflect very consistent performing. When searching for near matches, strings of short length (2-3 letters) are still likely to be similar to too many passages in the performance and hence not show what we are searching for. The problem with short sequences is that many of them are not distinctive enough to characterize a *single* musical idea or phrase, and therefore can be found in more than one context.

We are not interested in finding sequences of only two letters. As a consequence, we terminate the segmentation when the fittest subsequence we are able to find represents a string of only two characters. However, increasing δ encourages the EA to select longer strings – the δ value can be regarded as a fitness bonus per letter in the strings under consideration.

But finding longer matches is of course not a goal on it's own. We want to select the δ value and the threshold in such a way that sufficiently similar strings are accepted and too different ones rejected. We would like to draw this line where the strings found similar are as consistent as possible, i.e., located where the music is similar. Selecting the parameters which result in the lowest F-measure gives us a best possible segmentation where the similar strings found have the highest degree of consistency. This approach is described next.

5. Experiments

Using the F-measure as a consistency measure, we can run the EA with different parameter settings and evaluate the segmentations. Since the search algorithm is nondeterministic, it is necessary to run every experiment more than once in order to be certain that a segmentation was not just an occurrence of bad or good luck.

5.1. *Finding an F-measure optimal segmentation*

To find the δ value that results in the lowest F-measure we used the brute force approach. For 35 different values of δ every performance was segmented with the EA, and every experiment was repeated 10 times. The population of the EA was set to 100 individuals, and the EA was given 600 iterations for discovering new patterns. The δ value was gradually increased from 0.01 to 0.35 in steps of 0.01.

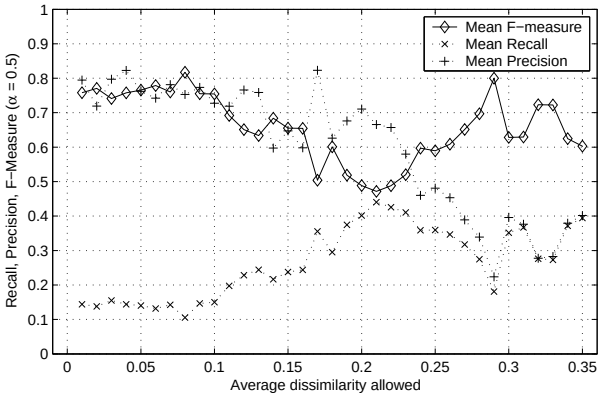

Fig. 5. Finding optimal parameters for segmenting the performance by Leonskaja. The points plotted represent the average value over 10 runs with each δ value.

Figure 5 shows for each value of δ the average F-measure, precision, and recall value calculated over the 10 segmentations with the EA of the performance by Leonskaja.

Allowing only little dissimilarity makes the algorithm behave in a conservative way – in a run with $\delta = 0.1$ only four strings were found with a total of 32 letters, but 26 of them being consistent. When δ is above 0.3, the segmentation is dominated by a few, but very long strings covering almost every letter in the string, not discriminating very well the sections in the music. The best average F-measure was obtained with $\delta = 0.21$. A segmentation in this setting found five categories of repeated strings of length 4 to 18 (see Table 3). Even though the strings may seem very different, the number of true positive matches of the letters in Table 3 was 80 and the number of false positives 32, giving a recall of 0.491, precision of 0.714 and F-measure of 0.418.

Table 3. A segmentation of the performance by Leonskaja.

Iteration (Length)	Start pos.	Strings found similar	Iteration (Length)	Start pos.	Strings found similar
1	3	DVJRIKRLJPJVDBCUCC	4	142	CPRTGHHJ
(18)	111	DQJROBTQJPJVJIDQCC	(8)	150	CPVOHHQJ
2	22	VNTJCPNJ	5	57	RNFX
(8)	38	UTJJQSNJ	(4)	62	MOJP
	134	VNRDCPQJ		66	MTGN
3	78	CBCUIR		94	NSGS
(6)	86	CBCUIR		159	VQGT
				164	RTGR

The strings from iteration 2 were found in three occurrences, plotted in Figure 6. Two of them refer to similar phrases, and the last (starting at pos. 134) to another phrase (although some resemblance can be argued). These three strings thus contribute 16 TPs and 8 FPs. It looks as if Leonskaja is more consistent in the loudness

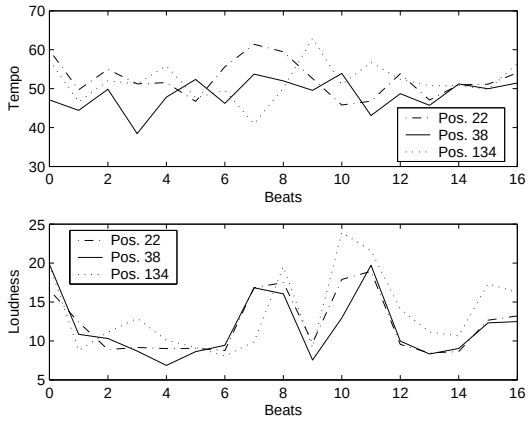

Fig. 6. The patterns starting at positions 22 (`VNTJCPNJ`) and 38 (`UTJJQSNJ`) refer to similar music; the music at pos. 134 (`VNRDCPQJ`) is somewhat different.

domain than in the tempo domain when playing this repeated phrase. The patterns found in iterations 1 and 3 are also applied to similar phrases.

5.2. *Ranking the performances*

It turns out that the performances have different F-measure optimal parameter settings – reflecting the degree of variance in the performance. We would like to compare the consistency in the performances by finding the individually optimal δ value for each pianist and then compare the respective lowest F-measure values. This is a fair comparison since every performance is characterized in the most consistent way possible.

By performing the experiment described in section 5.1 for every performance, we can now rank the performances according to the average best F-measure found when trying the 35 different δ values. The ranking is shown in Table 4.

This suggests that Barenboim and Horowitz are the most consistent performers of this piece. A Horowitz performance was segmented with the overall single best F-measure of 0.309. The segmentation of the performance by Lipatti gave the highest precision, but a mediocre recall results in a lower ranking. The ranking is not to be taken too literally – the standard deviation values shown indicate uncertainties in the ranking.

Gulda stands out by receiving the lowest ranking. Often three patterns are found in his performance – one of which is the largest source of confusion. It is a four letter pattern and occurs 10 times, where only 2 refer to similar phrases. Figure 7a) shows the 10 sequences found similar, plotted in the loudness space. It looks as if Gulda is not phrasing the music in longer sections. Certainly he does not play distinctively enough for the phrases to be recognized. Figure 7b) on the other hand shows a beautiful example of consistent music performance: Horowitz playing the beginning

Table 4. Ranking the pianists according to consistency. The ranking is done based on the lowest average F-measure over 10 runs for every of the 35 values of δ

Rank	δ	Recall	Precision	F-measure	St. dev F-m	Pianist
1	0.21	0.613	0.725	0.336	0.000	Barenboim
2	0.21	0.538	0.765	0.368	0.091	Horowitz
3	0.19	0.497	0.816	0.383	0.029	Lipatti
4	0.26	0.529	0.698	0.399	0.033	Maisenberg
5	0.28	0.589	0.578	0.416	0.000	Zimerman
6	0.21	0.440	0.666	0.472	0.023	Leonskaja
7	0.27	0.488	0.570	0.475	0.080	Uchida
8	0.19	0.388	0.805	0.478	0.057	Rubinstein
9	0.21	0.459	0.606	0.478	0.084	Kempff
10	0.22	0.452	0.583	0.491	0.048	Brendel
11	0.26	0.424	0.475	0.553	0.086	Pires
12	0.24	0.266	0.403	0.681	0.058	Gulda

of the piece compared to when he plays the repeat of the beginning.

When listening to Gulda and Horowitz the authors find that concerning tempo, Horowitz sounds like having a large 'momentum' behind the accelerandos and ritardandos – no sudden changes. Gulda on the other hand is much more vivid, taking fast decisions in tempo changes. This might account for some of the difference in consistency measured.

The question is what musical relevance there is in a consistency measure. It is measurable, and to some extent audible, but what does it mean? Some would argue that consistency is boring and predictable, but others that the predictability aids a better understanding of the music. Some would say that inconsistency has a flavor of uncertainty and randomness, but others that this makes the music varied,

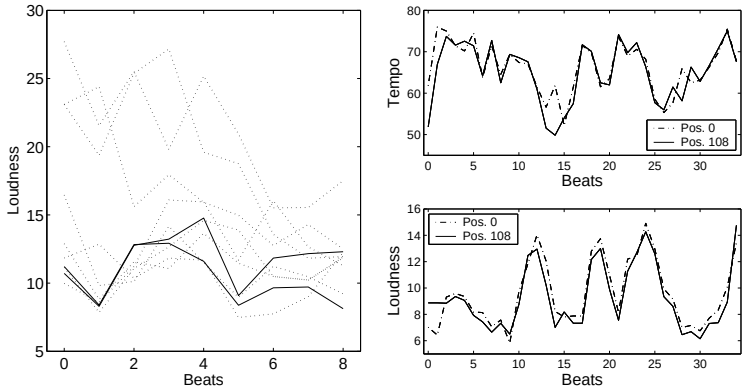

Fig. 7. a) Gulda playing a short pattern in 10 different variants (loudness plotted only). The two 'consistent' performances are intensified. b) Horowitz playing a long pattern in 2 very similar ways (tempo and loudness plotted separately): `FNLLIJPTGRGIRONOH` at pos. 0 and `FNMLGJROGRGHRLGOH` at pos. 108.

surprising and interesting.

5.3. *Finding similarities between the performers*

Our second application of the search algorithm is to find similar strings across all performances. This will reveal similarities in the playing style of different pianists.

For this experiment, we have incorporated in the fitness function a lookup table over phrase boundaries as represented in the analysis of the piece. Strings that agree with boundaries (starts and/or ends at subsection) are given a better fitness than strings not agreeing. This was done in a restrictive way: the fitness of an individual whose subsequences do not correspond with these boundaries is punished with the value 0.1 per non-agreeing subsequence 'end' (a total of 0.4 possible). The threshold was again set to zero, and δ fixed to 0.19. This value was the smallest δ value overall able to produce a F-measure optimal segmentation (see Table 4). The segmentation is therefore expected to be conservative. The task for the EA is now to find as many similar strings across all the 12 performance input strings that are within the threshold. By imposing this extra constraint we can force the EA to select equally bounded patterns across the performances. This eases the readability of the result. The result may however not be telling the whole truth.

Running the EA with a population of 300 individuals we obtain 12 segmentations shown in Figure 8. Similar substrings found (patterns) are indicated with boxes, with a number identifying the type. Above and below the strings, the letter position numbers are printed.

Similarities in performances can now be viewed as vertically aligned 'boxes' having the same identifier. For example the pattern labeled '1' is found 6 times at two different positions. This indicates that Barenboim, Horowitz, Uchida and Zimerman (pianists 0, 3, 10 and 11 in the figure) play the beginning of the piece in similar ways, and Barenboim and Uchida (pianist 0 and 11) also play the recapitulation (pos. 108) in this way.

The patterns '18', '13' and '9' are also found at these positions. These patterns were each played by different pianists suggesting individual interpretations by Brendel, Lipatti and Pires.

The pattern '17' represent similar ways of playing the characteristic four bars starting at pos. 77. The music is repeated at pos. 85. Barenboim, Brendel and Leonskaja seem to agree on interpreting this bit.

This segmentation displays a handful of similarities across the performances, but maybe to a higher degree within each performance. By loosening the similarity criteria, we are likely to find more patterns and more occurrences of the patterns, but are also introducing more errors.

6. Conclusion

We saw that a rather crude representation of the complex phenomenon of music performance, combined with an evolutionary search algorithm, can be used to re-

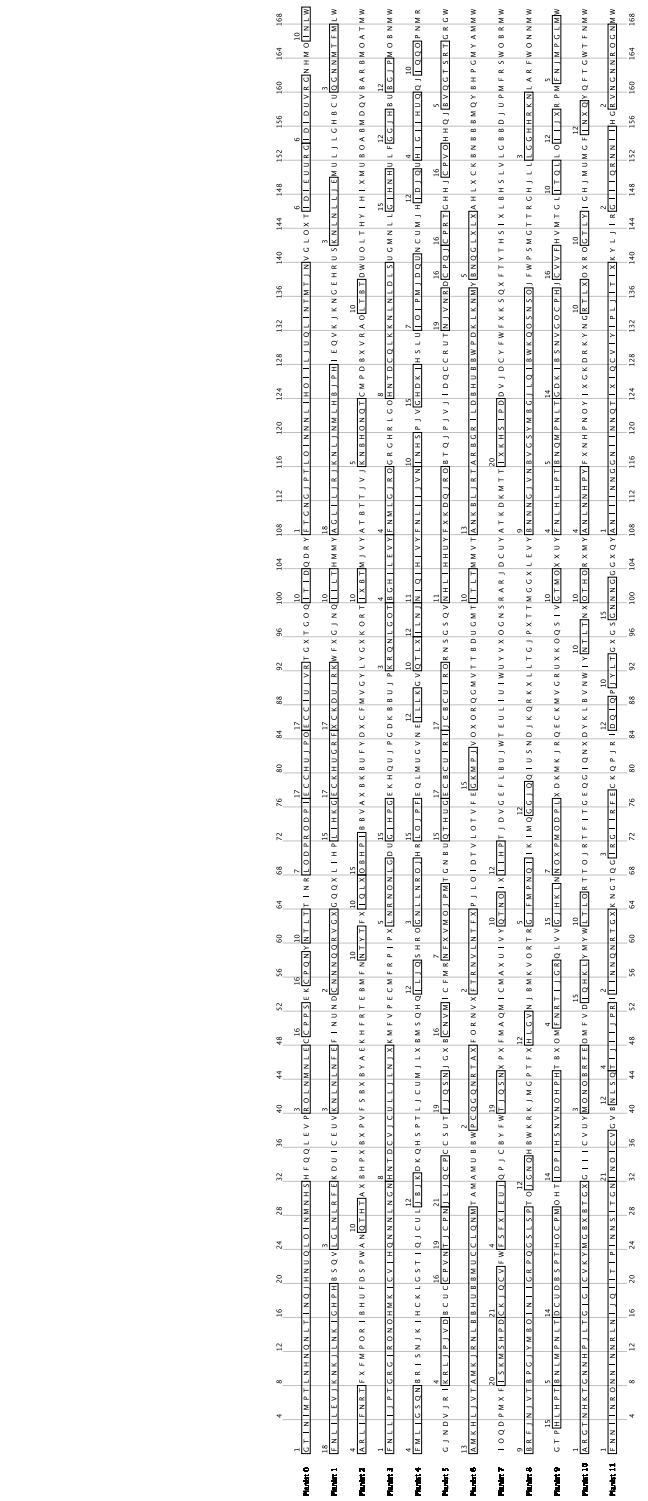

Fig. 8. All performances segmented in parallel

cognize patterns in performances of piano music. On the one hand, this exemplifies once more how music can be a valuable source of challenging problems for AI. On the other, this is another instance of AI making new and relevant contributions to the field of music performance research (other instances are e.g. [6,16]).

In terms of revealing a possible *Horowitz factor* our experiment suggested that consistency indeed could have something to do with it. However only experiments with a single piece of music have been conducted. We plan to continue this work with a larger corpus of more diverse musical material (though deriving precise measurements of expression in audio recordings is a very tedious task), in order to provide a deeper analysis of the musical meaning and significance of the results.

Acknowledgments

This research was supported by the Austrian FWF (START Project Y99) and the Viennese Science and Technology Fund (WWTF, project CI010). The Austrian Research Institute for AI acknowledges basic financial support from the Austrian Federal Ministries of Education, Science and Culture and of Transport, Innovation and Technology.

References

1. A. Gabrielsson. Music Performance. In Diana Deutsch, editor, *Psychology of Music*, pages 501–602. Academic Press, San Diego, 2nd edition, 1999.
2. R. L. de Mántaras and J. L. Arcos. AI and music: From composition to expressive performances. *AI Magazine*, 23(3):43–57, 2002.
3. G. Widmer, S. Dixon, W. Goebl, E. Pampalk, and A. Tobudic. In Search of the Horowitz Factor. *AI Magazine*, 24(3):111–130, 2003.
4. E.F. Clarke. Rhythm and timing in music. In D. Deutsch, editor, *The Psychology of Music*, pages 473–500. Academic Press, San Diego CA, 1999.
5. B. Repp. Diversity and commonality in music performance: An analysis of timing microstructure in Schumann's "Träumerei". *J. Acoust. Soc. Am.*, 92(5):2546–68, 1992.
6. W. Goebl, E. Pampalk, and G. Widmer. Exploring expressive performance trajectories. In *Proceedings of the 8th International Conference on Music Perception and Cognition (ICMPC'04)*, Evanston, IL, 2004.
7. S. Dixon. Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1):39–58, 2001.
8. S. Dixon. An interactive beat tracking and visualisation system. In *Proceedings of the International Computer Music Conference*, pages 215–218. La Habana, Cuba, 2001.
9. E. Zwicker and H. Fastl. *Psychoacoustics: Facts and Models*. Springer, Berlin, 1999. Second Edition.
10. J. Langner and W. Goebl. Visualizing expressive performance in tempo-loudness space. *Computer Music Journal*, 27(4):69–83, 2003.
11. S. Dixon, W. Goebl, and G. Widmer. The Performance Worm: Real time visualisation of expression based on langner's tempo-loudness animation. In *Proceedings of the International Computer Music Conference (ICMC), Göteborg,Sweden*, pages 361–364, 2002.
12. G. Widmer and P. Zanon. Automatic recognition of famous artists by machine. In

Proceedings of the 16th European Conference on Artificial Intelligence(ECAI'2004), Valencia, Spain, 2004.

13. C. Saunders, D. Hardoon, J. Shawe-Taylor, and G. Widmer. Using string kernels to identify famous performers from their playing style. In *Proceedings of the 15th European Conference on Machine Learning (ECML'2004)*, Pisa, Italy, 2004.
14. C. J. van Rijsbergen. *Information Retrieval*. Butterworth, London, 1979.
15. C.G. Nevill-Manning and I.H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.
16. G. Widmer. Discovering simple rules in complex data: A meta-learning algorithm and some surprising musical discoveries. *Artificial Intelligence 146(2), 129-148*, 2003.