Evolutionary Optimization of Music Performance Annotation

Maarten Grachten, Josep Lluís Arcos, and Ramon López de Mántaras

IIIA, Artificial Intelligence Research Institute,
CSIC, Spanish Council for Scientific Research,
Campus UAB, 08193 Bellaterra, Catalonia, Spain,
{maarten,arcos,mantaras}@iiia.csic.es,
http://www.iiia.csic.es

Abstract. In this paper we present an enhancement of edit distance based music performance annotation. The annotation captures musical expressivity not only in terms of timing deviations but also represents e.g. spontaneous note ornamentation. To reduce the number of errors in automatic performance annotation, some optimization is essential. We have taken an evolutionary approach to optimize the parameter values of cost functions of the edit distance. Automatic optimization is desirable since manual parameter tuning is unfeasible when more than a few performances are taken into account. The validity of the optimized parameter settings is shown by assessing their error-percentage on a test set.

1 Introduction

Although the use of the edit distance [7] is well known in the field of melodic similarity [8,12], score following/automatic accompaniment [3,10] and performance transcription [6,9], not much attention has been paid to its value for the expressive analysis and annotation of musical performances. The optimal alignment between score and performance does not only reveal timing deviations of performed notes, but (depending on the set of edit operations) conveys a much richer set of expressive variations, such as ornamentations, and fragmentations/consolidations. In the context of the *ProMusic* project[1] we are developing *Tempo Express*, a Case Based Reasoning system for applying tempo transformations to audio recordings of solo performances of jazz melodies [5]. In this system, we use the alignment information to automatically annotate performances [1]. The performance annotations serve as example cases to transform a performance for a given melody. As a result, the expressiveness of the transformed performance is not restricted to timing variations, but it can also contain for example ornamentations.

For a correct detection of phenomena such as ornamentations, fragmentations, and consolidations of notes using the edit distance, it is important to

[1] MCyT. TIC2003-07776-C2-02

assign appropriate costs to each of the edit operations. Since it turned out to be unfeasible to manually tune the costs to obtain correct annotations for a large set of performances, we tried to find good costs using a genetic algorithm. In this paper, we describe our experiments and results.

In section 2, we explain the idea of annotating performances by *performance events*. We wil show how performance annotations can be constructured using the edit distance algorithm, and motivate the chosen set of cost functions for assessing the costs of the edit operations. In section 3, we report how the parameter values in the cost functions were estimated, using a genetic algorithm, and evaluate the quality of the estimations. Conclusions are presented in section 4.

2 Performance Annotation

It has been widely acknowledged that human performances of musical material are virtually always quite different from mechanical renderings of the music. These differences (the musical expressivity) are thought to be vital for the aesthetic quality of the performance, and therefore it is worthwhile to have ways of making explicit the quality and quantity of these differences. The majority of research concerning musical expressivity is focused on the temporal, or dynamic variations of the notes of the musical score as they are performed [2,4,11,13]. In this context, the spontaneous insertions or deletions of notes by the performer are often discarded as artifacts, or performance errors. This may be due to the fact that most of this research is focused on the performance practice of classical music, where the interpretation of notated music is rather strict. Contrastingly, in jazz music performers often favor a more liberal interpretation of the score, so that expressive variation is not limited to variations in timing of score notes, but also comes in the form of e.g. deliberately inserted and deleted notes. Thus, research concerning expressivity in jazz music should pay heed to these phenomena and in addition to capturing the temporal/dynamical variations of score notes, the musical behavior of the performer should be described in terms of note insertions/deletion/ornamentations etcetera. One way to do this is to define these expressive phenomena as *performance events*, and then annotate the performance with a sequence of such events.

In the next subsections, we propose a set of performance events to be used in the performance annotation of saxophone jazz performances, we show how the edit distance algorithm can be used to construct the annotation, and propose cost functions to be used in the edit distance computation.

2.1 Choice of Performance Events

The decision which performance events to define is important, since they are in a sense the 'vocabulary' we use to represent the musician's performance behavior. The set of performance events proposed here (which is a slight extension of the one proposed in Arcos et. al. [1]), is chosen to reflect the variety of phenomena that we have actually encountered in a set of saxophone jazz performances.

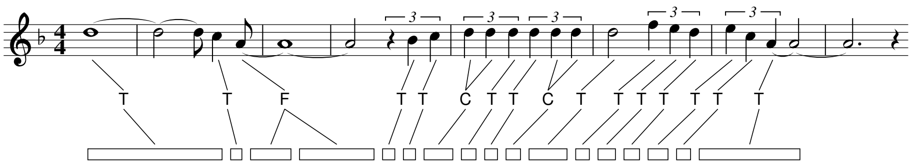

Fig. 1. performance annotation of the first phrase of *Once I Loved*, played by a saxophone at 220 bpm. The bars below denote the played notes (the bar lengths are representative for the note durations). The annotation is the sequence of performance events in the middle. 'T' is for transformation (of duration and onset), 'F' for fragmentation, and 'C' for consolidation

Based on the fact that the phrases in our data set were played by a professional musician and they were performed with the intention of giving a neutral interpretation of the score, it may be thought that expressive deviations of the score other than changing the timing or dynamics of the notes will occur only very infrequently, and that most of the performances can be represented by events that just describe how the timing and duration of score notes was changed as they were performed – i.e. *transformation* events. But listening to the performances revealed that other types of events occurred frequently as well. For example, some cases of note deletions and insertions were found. Apart from real insertions of notes, that gave the impression of an elaboration of the melody (such insertions occurred, but were rare), another type of insertion was found to occur rather often: ornamentation. By ornamentation we refer to one or more very short notes (typically about 100 or 200 ms.) that are usually a chromatic approach from below to the next score note. We have found such ornamentations to consist of one, two or three notes. Furthermore, we observed that consolidation (as described in the previous section) occurred in some performances. Occasionally, we found cases of fragmentation. Other transformations, such as sequential transposition (reversal of the temporal order of notes) were not encountered.

For illustration, figure 1 shows the annotation of a melodic phrase from the song 'Once I Loved' (A.C. Jobim). The performer fragmented the third note into two shorter notes, and in the repetition of triplet notes, there are two cases of consolidations. The other notes were played as is, with a lesser or greater degree of deviation in onset time and duration.

The kinds of events mentioned above can be visualized in a class hierarchy (as in figure 2) to make explicit their characteristics. The core idea of a performance event is that it relates the notes that are actually played by the performer to the notes that are written in the score. As such, every event refers either to one or more notes in the score, or to one or more notes in the performance, or both. We can distinguish, within this general class of *reference* events, those that refer to notes in the score and those that refer to elements in the performance. *Deletion* events refer to notes of the score that are not present in the performance (i.e. the notes that are not played), therefore they can be classified

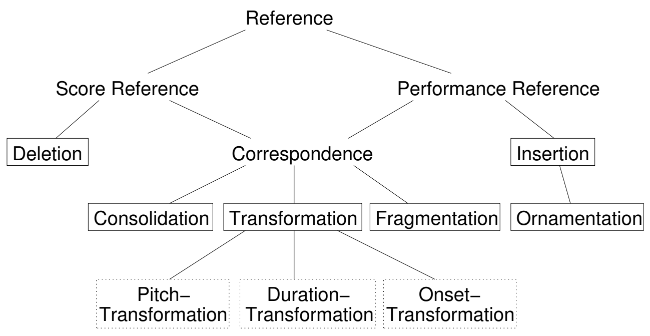

Fig. 2. A hierarchical representation of performance events for performance annotation. The unboxed names denote abstract classes; the boxed names denote 'concrete' classes that are used in the performance annotation. The dottedly boxed names denote classes that are derived from the concrete classes

as *score-reference* events. Conversely, *insertion* events refer only to elements in the performance (i.e. the notes that were added), so they form a subclass of *performance-reference* events. *Transformation*, *consolidation* and *fragmentation* events refer to elements from both the score and the performance and thus form a shared subclass of *score-reference* and *performance-reference* events. We call this class *correspondence* events.

The *reference*, *score-reference*, *performance-reference* and *correspondence* classes are abstract classes that are just conceived to express the relationships between concrete classes of events, and are not intended to be used directly in the performance annotation. The concrete classes, that are used to construct the performance annotation, are depicted in figure 2 with boxes. They are:

Insertion Represents the occurrence of a performed note that is not in the score
Deletion Represents the non-occurrence of a score note in the performance
Consolidation Represents the agglomeration of multiple score notes into a single performed note
Fragmentation Represents the performance of a single score note as multiple notes
Transformation Represents the change of nominal note features
Ornamentation Represents the insertion of one or several short notes to anticipate another performed note

In the case of *transformation*, we are not only interested in the one-to-one correspondence of performance elements to score elements itself, but rather in the changes that are made to attribute values of score notes when they are transformed into performance elements. Therefore, we view transformation events as compositions of several transformations, e.g. *pitch transformations*, *duration transformations* and *onset transformations*.

2.2 Constructing Performance Annotations Using Edit Distance

Performance annotations, being sequences of performance events, can be treated as a sequence of operations that tell you how to perform a written score, or alternatively, how to transform a sequence of score notes into a sequence of performed notes. As such, the relation between the performance annotation and the concept of *optimal alignment* between score and performance becomes obvious. When performance events are defined as edit operations, then computing the edit distance between the score and the performance yields a sequence of edit operations, from which the performance annotation is constructed.

One problematic aspect of defining the performance events as edit operations, is the fact that the subclasses of *transformation* events (*pitch transformations*, *duration transformations* and *onset transformations*), can occur simultaneously (that is, they can refer to the same score and performance notes), whereas in the edit distance, each sequence element is covered by exactly one operation. Our solution is to have a single Transformation operation for the computation of the alignment, as a rough identification of the expressivity. In a second stage, after the alignment has been computed, the score and performance events corresponding to Transformation operations can be compared in more detail to establish which of the pitch, duration, and onset transformation really occurred. The corresponding classes are shown in figure 2 as dotted boxes.

The edit distance between a source and a target sequence is defined as the minimum cost of transforming the source sequence into the target sequence using a fixed set of edit operations. This cost can be calculated using the following recurrence equation, that defines the distance $d_{m,n}$ between two sequences $\langle a_1, a_2, ..., a_m \rangle$ and $\langle b_1, b_2, ..., b_n \rangle$ (using insertion, deletion and replacement, the standard set of edit operations):

$$d_{i,j} = min \begin{cases} d_{i-1,j} + w(a_i, \emptyset) & \text{(deletion)} \\ d_{i,j-1} + w(\emptyset, b_j) & \text{(insertion)} \\ d_{i-1,j-1} + w(a_i, b_j) & \text{(replacement)} \end{cases}$$

for all $0 \leq i \leq m$ and $0 \leq j \leq n$, where m is the length of the source sequence and n is the length of the target sequence. The initial conditions for the recurrence equation are:

$$\begin{aligned} d_{i,0} &= d_{i-1,j} + w(a_i, \emptyset) && \text{(deletion)} \\ d_{0,j} &= d_{i,j-1} + w(\emptyset, b_j) && \text{(insertion)} \\ d_{0,0} &= 0 \end{aligned}$$

The weight function w, defines the cost of operations, such that e.g. $w(a_4, \emptyset)$ returns the cost of deleting element a_4 from the source sequence, and $w(a_3, b_5)$ returns the cost of replacing element a_3 from the source sequence by the element b_5 of the target sequence.

For two sequences a and b, consisting of m and n elements respectively, the values $d_{i,j}$ (with $0 \leq i \leq m$ and $0 \leq j \leq n$) are stored in an $n+1$ by $m+1$ matrix. The value in the cell at the lower-right corner, $d_{m,n}$ is taken as the distance between a and b, that is, the minimal cost of transforming the sequence $\langle a_0, ..., a_m \rangle$ into $\langle b_0, ..., b_n \rangle$.

In our particular case, we define the following edit operations: insertion, deletion, ornamentation, transformation, fragmentation, and consolidation. To use these operations for score performance alignment, cost values must be assigned to each of them. This is done by means of weight functions for each type of operation (w in the recurrence equation).

2.3 The Cost Values

Ideally, the cost values for each type of operation will be such that the resulting optimal alignment corresponds to an intuitive judgment of how the performance aligns to the score (in practice, the subjectivity and ambiguity that is involved in establishing this mapping by ear, turns out to be largely unproblematic). The main factors that determine which of all the possible alignments between score and performance is optimal, will be on the one hand the features of the note elements that are involved in calculating the cost of applying an operation, and on the other hand the relative costs of the operations with respect to each other.

In establishing which features of the compared note elements are considered in the comparison, we have taken the choices made by Mongeau and Sankoff [8] as a starting point. In addition to pitch and duration information (proposed by Mongeau and Sankoff), we have decided to incorporate the difference in position in the costs of the correspondence operations (transformation, consolidation and fragmentation), because this turned out to improve the alignment in some cases. One such case occurs when one note in a row of notes with the same pitch and duration is omitted in the performance. Without taking into account positions, the optimal alignment will delete an arbitrary note of the sequence, since the deletions of each of these notes are equivalent based on pitch and duration information only. When position *is* taken into account, the remaining notes of the performance will all be mapped to the closest notes in the score, so the deletion operation will be performed on the score note that remains unmapped, which is often the desired result.

It is important to note that when combining different features, like pitch, duration and onset into a cost-value for an operation, the relative contribution of each term is rather arbitrary. For example when the cost of transforming one note into another would be defined as the difference in pitch plus the difference in duration, the outcome depends on the units of measure for each feature. The relative weight of duration and pitch is not the same when measured in seconds, as when measured in beats. Similarly, pitch could be measured in frequency, semitones, scale steps, etcetera. Therefore, we have chosen a parametrized approach, in which the relative contribution of each term in the weight function is weighted by a constant parameter value.

The other aspect of designing cost-functions is the relative cost of each operation. After establishing the formula for calculating the weights of each operation, it may be that some operations should be systematically preferred to others. This independence of costs can be achieved by multiplying the cost of each operation by a factor and adding a constant.

The cost functions w for the edit operations are given below. The arguments of the functions are elements from a sequence of score notes s, and a sequence of performed notes p. $\mathcal{P}$, $\mathcal{D}$, and $\mathcal{O}$ are functions such that $\mathcal{P}(x)$ returns the pitch (as a MIDI number) of a score note or performed note x, $\mathcal{D}(x)$ returns its duration, and $\mathcal{O}(x)$ returns its onset time. Equations 1, 2, 3, 4, 5, 6 define the costs of deletion, insertion, ornamentation, transformation, consolidation and fragmentation, respectively.

$$w(s_i, \emptyset) = \alpha_d \cdot \mathcal{D}(s_i) \tag{1}$$

$$w(\emptyset, p_j) = \alpha_i \cdot \mathcal{D}(p_j) \tag{2}$$

$$w(\emptyset, p_j, \cdots, p_{j+L+1}) = \alpha_o \cdot \begin{pmatrix} \beta \cdot \sum_{l=1}^{L} |1 + \mathcal{P}(p_{j+l}) - \mathcal{P}(p_{j+l-1})| + \\ \gamma \cdot \sum_{l=0}^{L} \mathcal{D}(p_{j+l})\, | \end{pmatrix} \tag{3}$$

$$w(s_i, p_j) = \alpha_t \cdot \begin{pmatrix} \beta\cdot\, |\mathcal{P}(s_i) - \mathcal{P}(p_j)\,| + \\ \gamma\cdot\, |\mathcal{D}(s_i) - \mathcal{D}(p_j)\,| + \\ \delta\cdot\, |\mathcal{O}(s_i) - \mathcal{O}(p_j)\,| \end{pmatrix} \tag{4}$$

$$w(s_i, ..., s_{i+K}, p_j) = \alpha_c \cdot \begin{pmatrix} \beta \cdot \sum_{k=0}^{K} |\mathcal{P}(s_{i+k}) - \mathcal{P}(p_j)\,| + \\ \gamma\cdot\, |\mathcal{D}(p_j) - \sum_{k=0}^{K} \mathcal{D}(s_{i+k})\,| + \\ \delta\cdot\, |\mathcal{O}(s_i) - \mathcal{O}(p_j)\,| \end{pmatrix} \tag{5}$$

$$w(s_i, p_j, ..., p_{j+L}) = \alpha_f \cdot \begin{pmatrix} \beta \cdot \sum_{l=0}^{L} |\mathcal{P}(s_i) - \mathcal{P}(p_{j+l})\,| + \\ \gamma\cdot\, |\mathcal{D}(s_i) - \sum_{l=0}^{L} \mathcal{D}(p_{j+l})\,| + \\ \delta\cdot\, |\mathcal{O}(s_i) - \mathcal{O}(p_j)\,| \end{pmatrix} \tag{6}$$

The parameters β,γ, and δ control the influence of pitch, duration, and onset, respectively. α_d, α_i, α_o, α_t, α_c, and α_f are the parameters that scale the costs of deletion, insertion, ornamentation, transformation, consolidation and fragmentation, respectively.

The costs of transformation (4), consolidation (5), and fragmentation (6), are principally constituted of the differences in pitch, duration and onset times between the compared elements. In the case of one-to-many matching (fragmentation) or many-to-one (consolidation), the difference in pitch is calculated as the sum of the differences between the pitch of the single element and the pitches of the multiple elements. The difference in duration is computed between the duration of the single element and the sum of the durations of the multiple elements. The difference in onset is computed between the onset of the single element and the onset of the onset of the first of the multiple elements. The cost of deletion (1) and insertion (2)is determined by the duration of the deleted element. The cost of ornamentation (3) is determined by the pitch relation of the ornamentation elements and the ornamented element (chromatically ascending sequences are preferred), and the total duration of the ornamentation elements.

3 Experimentation

The introduction of the nine parameters in the cost functions comes with the problem of finding appropriate values for those parameters. Although the edit distance has some robustness (it aligns sequences reasonably well, even if bad parameter values are chosen), it is difficult to bring the amount of annotation errors down to a few percent. Manually tuning the parameters is possible for a small set of performances, but this becomes unfeasible for larger sets (adjustments that improve the annotation of one performance, worsened the annotation of others). Surprisingly, our manually tuned settings hardly improved the accuracy of annotation with respect to random parameter settings when tested on larger sets of performances. Therefore, we have employed a genetic algorithm to obtain a good parameter setting. In this section we describe our experimentation with the tuning of the parameters.

The idea of the evolutionary optimization of the parameter values is rather simple: an array of the nine parameter values (one value for each parameter) can be treated as a chromosome. The number of errors produced in the annotation of a set of performances using that set of parameter values, is inversely related to the fitness of the chromosome. By evolving an initial population of (random) chromosomes through crossover, mutation and selection, we expect to find a set of parameter values that minimizes the number of annotation errors, and thus improves automatic performance annotation.

We are interested in two main questions. The first is whether it is possible to find a parameter setting that works well in general. That is, can we expect a parameter setting that worked well for a training set to perform well on unseen performances? The second question is whether there is a single setting of parameter values that optimizes the annotations. It is also conceivable that good annotations can be achieved by several different parameter settings.

3.1 Experiment Setup

We have run the genetic algorithm with two different (non-overlapping) training sets, both containing twenty performances. These were (monophonic) saxophone performances of eight different phrases from two jazz songs (*Body and Soul*, and *Once I Loved*), performed at different tempos. For each of the performances, the correct annotation was available. The fitness of the populations was assessed using these annotations.

The fitness evaluation of a population (consisting of 20 chromosomes) on the training set is a rather time consuming operation. Therefore, it can take a long time before a good solution is obtained, starting the evolution with a randomly initialized population. In an attempt to solve this problem, we initialized the population with solutions that were trained on the individual phrases of the training set (which is a much faster procedure). Assuming that the solution optimized for one phrase may in some cases work for other phrases, this speeds up the time needed to find a good solution for the whole training set.

A new generation is generated from an old generation as follows: From the old generation (consisting of N chromosomes), the k best chromosomes are selected (where k is dependent on the distribution of the fitness across the population); Then, $N - k$ new chromosomes are created by a cross-over of the selected chromosomes; The newly generated chromosomes are mutated (multiplying each parameter value by a random value), and the $N - k$ mutated chromosomes, together with the n (unchanged) chromosomes from the old generation, form the new generation.

3.2 Fitness Calculation

The fitness of the chromosomes is calculated by counting the number of annotation errors using the parameter values in the chromosome. For example, assume that the correct annotation of a melodic fragment is 'T T C T', and the annotation of that fragment obtained by using the parameter values of the chromosome is 'T T T D T' (that is, a consolidation operation is confused with an transformation and a deletion operation). The 'C' doesn't match to an element in the second sequence, and the 'T' and 'D' don't match to elements in the first sequence and thus three errors occur. To count the errors between the correct and the predicted annotations (which are represented as sequences of symbols), we use the edit distance (don't confuse this use of the edit distance to *compare* annotations with the use of the edit distance to *generate* annotations).

For a given set S of performances (for which the correct annotations are known), we define the fitness of a chromosome c as:

$$\mathit{fit}(c) = \frac{1}{E(c, S) + 1}$$

where $E(c, S)$ is the total number of errors in the predicted annotations for S using the parameter values in c. The fitness function *fit* ranges from zero to one. Obviously, a fitness value of one is the most desirable, since it corresponds to zero annotation errors.

3.3 Results

For each of the two training sets, Tr1 and Tr2, the evolution algorithm was run three times. The resulting parameter settings are shown in figure 3. Table 1 shows the number of annotation errors each of the parameter settings produced on the training sets, and on a test set (a set of 35 performances, none of which occurred in Tr1 or Tr2). The average number of annotation errors on the test set is about 32 on a total of 875 annotation elements in the test set, an error-percentage of 3.66%. This is only slightly higher than the error-percentages on

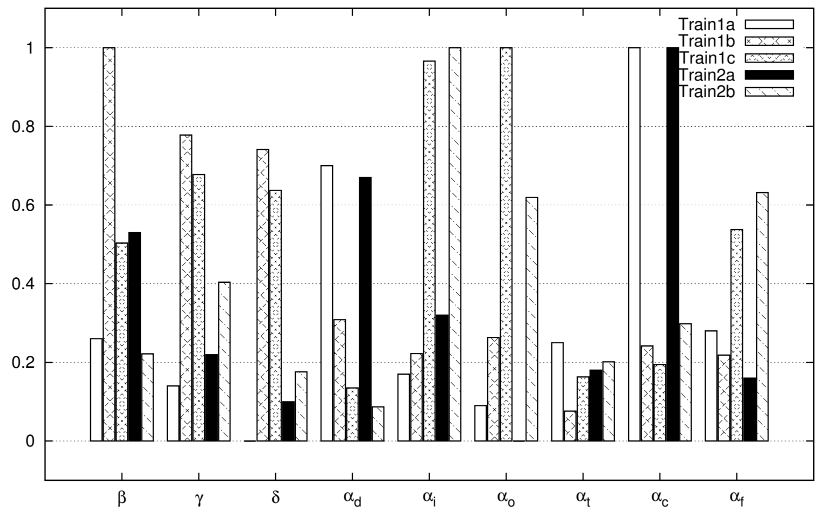

Fig. 3. Estimated parameter values for two different training sets (Tr1 and Tr2). Three runs were done for each set (a, b, and c). The x-axis shows the nine different parameters of the cost functions (see section 2.3). For each parameter the values are shown for each run on both training sets

the training sets: 2,60% for Tr1, and 2,37% for Tr2 (averaged over three runs), and substantially lower than the average error-percentage of random parameter settings on the test set, which is about 13,70%.

	Tr1a	Tr1b	Tr1c	Tr2a	Tr2b	Tr2c
Errors on Train	19 (3.89)	9 (1.84)	10 (2.05)	11 (2.30)	12 (2.51)	11 (2.30)
Errors on Test	19 (2.17)	26 (2.97)	30 (3.43)	19 (2.17)	32 (3.66)	65 (7.43)

Table 1. Annotation errors produced by the obtained solutions for three different runs (denoted by the letters a, b, and c) on two different training sets (Tr1 and Tr2) and a test set. The first row shows the number of errors on the set that the solutions were trained on, and the corresponding percentages in parentheses (Tr1 contained 488 annotation elements in total, and Tr2 contained 479). The second row shows the number of errors on the test set (875 elements), with percentages in parentheses

Table 2 shows the pair-wise correlations of the values. As can be seen from the cross-correlations in the table, the parameter settings did not all converge to the same values. Nevertheless, there were some cases in which the parameters were highly correlated. In particular the solutions found in runs Tr1a, and Tr2a are highly similar (this can be easily verified by eye in figure 3. A rather strong correlation is also observed between the solutions found in Tr1c and Tr2b, and those in Tr1b, and Tr2c. It is interesting that the correlated solutions were obtained using non-overlapping sets of performances. This is evidence that the solutions found are approximations of a single parameter setting that is valid for the performances in both training sets. In the case of the solutions of Tr1a and Tr2a, the approximated parameter setting may also have a more general validity, since both solutions have a low error number of annotations on the test set as well (see table 1).

	Tr1a	Tr1b	Tr1c	Tr2a	Tr2b	Tr2c
Tr1a	1.00	-0.32	-0.70	0.92	-0.32	-0.28
Tr1b	-0.32	1.00	0.17	-0.02	-0.33	0.68
Tr1c	-0.70	0.17	1.00	-0.61	0.76	0.07
Tr2a	0.92	-0.02	-0.61	1.00	-0.33	-0.12
Tr2b	-0.32	-0.33	0.76	-0.33	1.00	-0.47
Tr2c	-0.28	0.68	0.07	-0.12	-0.47	1.00

Table 2. Cross-correlations of the parameter values that were optimized using two different training sets (Tr1 and Tr2), and three runs for each set (a, b, and c)

4 Conclusions and Future Work

We have presented a method to enhance the automatic annotation of music performances. The annotation includes information about e.g. note ornamentations and deletions as part of the musical expressivity. To correctly detect such phenomena, an evolutionary approach was chosen to optimize the parameter values of cost functions, that were used in the (edit distance based) performance annotation process.

Two main questions we have tried to answer is whether it is possible to find a parameter setting that has a broader validity than just the set of performances it was optimized for, and whether there is a single parameter setting that optimizes the annotations. All solutions from different trials on two non-overlapping sets of performances substantially improved the quality of annotation of a test set over random parameter settings. Moreover, cross-correlations were found between some parameter settings that were optimized for different training sets. This suggests that they are approximations of a parameter setting that works well for a larger group of performances. In general however, the solutions did not all converge to a single set of parameter values.

In the future, we wish to extend the experiments to see whether the solutions found converge to a limited range of parameter settings. And if so, we wish to investigate how the distributions of values over the parameters relate to each other (for example, do high α_c values imply low γ and δ values and vice versa?).

Acknowledgments

This research has been partially supported by the Spanish Ministry of Science and Technology under the project TIC 2003-07776-C2-02 "CBR-ProMusic: Content-based Music Processing using CBR" and EU-FEDER funds. The authors acknowledge the Music Technology Group of the Pompeu Fabra University for providing the analysis module that was used to extract note information from the audio files.

References

1. J. Ll. Arcos, M. Grachten, and R. López de Mántaras. Extracting performer's behaviors to annotate cases in a CBR system for musical tempo transformations. In *Proceedings of the Fifth International Conference on Case-Based Reasoning (ICCBR-03)*, 2003.
2. Sergio Canazza, Giovanni De Poli, Stefano Rinaldin, and Alvise Vidolin. Sonological analysis of clarinet expressivity. In Marc Leman, editor, *Music, Gestalt, and Computing: studies in cognitive and systematic musicology*, number 1317 in Lecture Notes in Artificial Intelligence, pages 431–440. Springer, 1997.
3. R. Dannenberg. An on-line algorithm for real-time accompaniment. In *Proceedings of the 1984 International Computer Music Conference*. International Computer Music Association, 1984.
4. P. Desain and H. Honing. Does expressive timing in music performance scale proportionally with tempo? *Psychological Research*, 56:285–292, 1994.
5. E. Gómez, M. Grachten, X. Amatriain, and J. Ll. Arcos. Melodic characterization of monophonic recordings for expressive tempo transformations. In *Proceedings of Stockholm Music Acoustics Conference 2003*, 2003.
6. E. W. Large. Dynamic programming for the analysis of serial behaviors. *Behavior Research Methods, Instruments & Computers*, 25(2):238–241, 1993.
7. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.
8. M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.
9. B. Pardo and W. Birmingham. Improved score following for acoustic performances. In *International Computer Music Conference (ICMC)*, Goteborg, Sweden, 2002. The International Computer Music Association.
10. M. Puckette and A. C. Lippe. Score following in practice. In *Proceedings, International Computer Music Conference*, pages 182–185, San Francisco, 1992. International Computer Music Association.
11. B. H. Repp. Quantitative effects of global tempo on expressive timing in music performance: Some perceptual evidence. *Music Perception*, 13(1):39–58, 1995.
12. Ll. A. Smith, R. J. McNab, and I. H. Witten. Sequence-based melodic comparison: A dynamic programming approach. In W. B. Hewlett and E. Selfridge-Field, editors, *Melodic Similarity. Concepts, Procedures, and Applications*, Computing in Musicology, pages 101–118. MIT Press, 1998.
13. G. Widmer. Large-scale induction of expressive performance rules: First quantitative results. In *Proceedings of the International Computer Music Conference (ICMC2000)*, San Francisco, CA, 2000. International Computer Music Association.