Evolving Performance Models by Performance Similarity: Beyond Note-to-note Transformations

Amaury Hazan
Music Technology Group
Pompeu Fabra University
Ocata 1
08001 Barcelona, Spain
ahazan@iua.upf.edu

Maarten Grachten
Artificial Intelligence Research Institute
Spanish Council for Scientific Research
(IIIA - CSIC)
Campus UAB, 08193 Bellaterra, Spain
maarten@iiia.csic.es

Rafael Ramirez
Music Technology Group
Pompeu Fabra University
Ocata 1
08001 Barcelona, Spain
rramirez@iua.upf.edu

Abstract

This paper focuses on expressive music performance modeling. We induce a population of score-driven performance models using a database of annotated performances extracted from saxophone acoustic recordings of jazz standards. In addition to note-to-note timing transformations that are invariably introduced in human renditions, more extensive alterations that lead to insertions and deletions of notes are usual in jazz performance. In spite of this, inductive approaches usually treat these latter alterations as artifacts. As a first step, we integrate part of the alterations occurring in jazz performances in an evolutionary regression tree model based on strongly typed genetic programming (STGP). This is made possible (i) by creating a new regression data type that includes a range of melodic alterations and (ii) by using a similarity measurement based on an edit-distance fit to human performance similarity judgments. Finally, we present the results of both learning and generalization experiments using a set of standards from the *Real Book*.

Keywords: Evolutionary Modeling, Expressive Music Performance, Melodic Similarity

1. Introduction

Modeling expressive music performance is a challenging aspect for several research areas ranging from computer music to artificial intelligence and behavioral psychology. The focus of this paper is the study of how skilled musicians (saxophone jazz players in particular) express and communicate their view of the musical and emotional content of musical pieces by introducing deviations and changes of various parameters. Traditionally, the deviations introduced by the performer are studied on a note-to-note basis. However, jazz performance is characterized by a stronger manifestation of expressivity. A set of common deviations, which we call *performance events*, is listed below.

Insertion The occurrence of a performed note that is not in the score

Deletion The non-occurrence of a score note in the performance

Consolidation The agglomeration of multiple score notes into a single performed note

Fragmentation The performance of a single score note as multiple notes

Transformation The change of nominal note features like onset time, duration, pitch, and dynamics

Ornamentation The insertion of one or several short notes to anticipate another performed note

The listed performance events tend to occur persistently throughout different performances of the same phrase. Moreover, performances including such events sound perfectly natural, so much that it is sometimes hard to recognize them as deviating from the notated score. This supports our claim that even the more extensive deviations are actually a common aspect of (jazz) performance. As a starting point, we deal in this paper with the performance events that occur most commonly in our training database, namely transformations, consolidations and ornamentations. Finally, more subtle alterations such as intra-note modulations (e.g. vibrato or loudness shape, see [10]) play an important role in a saxophone rendition, however such alterations are out of the scope of this work. We focus here on the influence of the melodic context on the expressive gestures presented above. The problem can be stated as follows: how do underlying patterns in the score account for expressive gestures in a performance? Our approach is the following: we extract a set of features describing the melodic context of a given set of musical fragments, e.g. the melodic and rhythmic intervals between notes, their metrical position within the bar, or how they are perceptually grouped. On the other hand, we analyze the recordings of the corresponding pieces performed by a saxophone player. Each performance transcription can be stored and considered as training performance. Based on the melodic context features, we used each model to predict a set of performance events. A similarity assessment between training and predicted performances is then computed (see Section. 3) and used to guide the evolutionary process. Figure 1 summarizes this approach.

The modeling task lies in finding an appropriate mapping from melodic descriptors to expressive features that

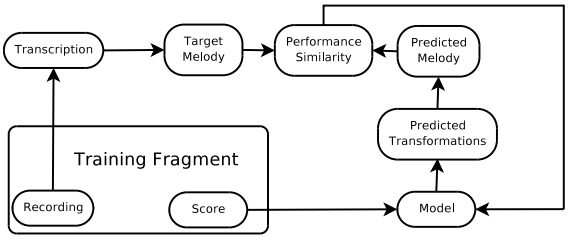

Figure 1. General framework for evolving a performance model. For each training fragments, we provide an acoustic recording of the performance and the corresponding score. The transcription of the recording is considered as target and is compared with the melody generated by the model. The computed similarity in used to characterize the fitness of the model in the evolutionary process

reflects patterns appearing in a training set. Moreover, we are interested in a mapping that shows good generalization capabilities i.e. that produces accurate predictions when processing unseen data. Despite of the challenge it represents for multidisciplinary research, inductive expressive performance modeling has motivated relatively few works. In [12, 14, 13] the authors investigate loudness and tempo variations of classical piano performances and induce multi-level performance models. In [4] the authors use a Case-Base Reasoning system to model expressivity-aware tempo transformations and introduce a framework to describe extensive melody alterations. In [9], we use Inductive Logic Programming to build a set of independent greedy regression tree models that can generate and explain expressive performances regarding timing, ornamentations and consolidations. A major drawback of this approach is that the different models are induced separately. As a consequence, the set of tree models may produce predictions that are not consistent. Here, we present a way to integrate *generative* predictions for a range of performance events into self-contained performance tree models. This is made possible by using an evolutionary regression tree approach ([5]). This technique is flexible enough (i) to produce *structured* predictions that are needed to integrate a number of possible performance events, and (ii) to let us define an accuracy measurement that goes beyond pair-wise comparisons.

1.1. Why use evolutionary computation techniques?

We present in this section the arguments that led us to consider evolutionary computation (EC), and more specifically strongly typed genetic programming (STGP, [8]), as convenient methods for building expressive performance models. Even if the aspects we present here are of interest for applying EC to a whole range of domains [6], we focus on how these techniques can benefit our particular problem.

1.1.1. Population of performance models

Human performance is an inexact phenomenon: even if performers learn through rehearsal to produce globally similar performances, they could hardly reproduce exactly identical renditions. This is particularly true in jazz music, where the improvisational contribution to a performance is fundamental. At each new rendition, small alterations in the timing and other expressive features of the musical piece are introduced. Expressive performance models should take into account this aspect. In contrast, traditional decision tree models are essentially deterministic. That is, once induced a feature-based decision tree model will produce a constant prediction for transforming a given fragment. EC provide an elegant way to cope with this limitation by evolving a population of models, with benefits for both end-users and researchers. The former have access to a population of models, each one which produce distinct set of performance events. The latter can refine specific design decisions by analyzing their consequence on the whole population.

1.1.2. Custom data types for structured modeling/prediction

Although it is possible to use evolutionary computation techniques in a straightforward way i.e. using templates and well-defined data types, the EC programmer may design custom data types for the model inputs and outputs. In the case of STGP, it is possible to refine the input/output specifications for each primitive of the genetic program. Here, we make use of such flexibility to define a structured prediction data type that takes into account features of note timing, ornamentations, consolidations. This is a first step towards a more complete prediction of performance events. We will present this prediction type in detail in the next section.

1.1.3. Integration of domain-specific knowledge

When modeling a given problem, it is sometimes crucial to integrate elements of domain-specific knowledge in the system. In the case of feature-based tree modeling, one can define this knowledge at several levels. First, the tree structure has to be defined so that it matches a regression tree structure. Also, one can easily handle the prior distribution of inputs and outputs by defining the initialization operator for each primitive in use in the genetic program. In our case, we refer to *constant generation*: on the one hand, features are compared to constants across the successive tests, these constants have to respect the statistical distribution of the feature space. On the other hand, single or even structured constant predictions are generated independently to where they will appear in a tree. Consequently, the role of the initialization operator is to provide a first guess for both features and prediction values. Finally, the most central aspect of EC is that the user has the freedom (and the responsibility) to define a fitness function that is suitable for a given problem. This enables us to include elaborate accuracy measurements such as performance similarity in the evolution process.

The rest of the paper is organized as follows: Section 2 presents the types, primitives, and operators of the tree model. We define the evolutionary constraints needed for

producing consistent regression trees from both typing and logical point of view, and define a prediction type for our problem. Section 3 presents a performance-similarity criterion based on edit-distance. In Section 4 we show the learning and generalization accuracy of the evolved models using an expressive performance database and discuss the results. Finally, Section 5 presents our conclusions as well as future extensions to this work.

2. Evolutionary regression trees

Regression tree models are a specific instance of tree programs. Their general structure is the following: Each node is a test comparing an input with a numerical constant (typically a *lower than* inequality). The outcome of a test is either a new test or a prediction. Furthermore, successive tests must be arranged to form consistent rules. We integrate these constraints in the evolutionary process. First, it is possible to ensure that the models will be structurally consistent by defining appropriate types and primitives. We then address the issue of logical consistency and define a mechanism to ensure it by overriding the evolutionary operators.

2.1. Types and primitives

Type consistency refers to how each primitive in the tree is connected to its neighbors. We use this STGP feature to ensure the desired regression tree structure. We define four different types: *InputValue*, *FeatValue*, *Bool* and *RegValue*. The first two types represent floating-point values, the first being used as terminal for the inputs of the model, the second one encapsulating constants to be compared with the inputs. The third type represents boolean values used as outcome of these tests.

The last type is used for encoding the model predictions. We give in table 1 details of the *RegValue* data type. The first three *RegValue* members, namely Duration Ratio, Onset Deviation and Relative Loudness, are float-valued and they refer to a transformation event. The fourth member, namely Ornamentation Event, is a boolean value indicating whether an ornamentation will be generated. If it is false, no ornamentation will be generated, and members 5 to 8 (namely Ornamentation Relative Onset, Ornamentation Absolute Duration, Ornamentation Relative Pitch and Ornamentation Relative Loudness) are ignored. Otherwise, we use these float-valued members to generate the ornamentation. The ninth *RegValue* member, namely Consolidation Event, is a boolean indicating whether a consolidation will be generated. Otherwise, the last Consolidation Relative Duration member is used along with the three first members to produce a consolidation. Note that this design allow the system to predict the following events combinations: transformation, ornamentation+transformation, consolidation, ornamentation+consolidation.

Corresponding to these types, we present in table 2 the needed primitives and show how they correspond to the structure of a regression tree. First, LT primitive tests whether an input of type *InputValue* is lower than a *FeatValue* typed constant generated by the EFV primitive. A LT primitive returns a *Bool* that is used as first argument of the IF primitive. The latter performs a test on this boolean value and returns a *RegValue*. If the test succeeds, the second argument is returned, otherwise, the third argument is returned. Both of these arguments are *RegValue* typed. Finally ERV primitive generates an *RegValue* prediction constant. As a result, the second and third arguments of an IF primitive can either be linked to an ERV primitive, or to another IF primitive, chaining successive tests until an ERV primitive is reached.

Table 1. *RegValue* **data type description**

Member Name	Type
Duration Ratio	float
Onset Deviation	float
Relative Loudness	float
Ornamentation Event	bool
Ornamentation Relative Onset	float
Ornamentation Absolute Duration	float
Ornamentation Relative Pitch	float
Ornamentation Relative Loudness	float
Consolidation Event	bool
Consolidation Relative Duration	float

Table 2. STGP primitives in use for this work. Each primitive is presented along with the number of arguments it handles, the type of each argument, and its return type; Note that primitives EFV and ERV, which are used for constant generation, take no argument as input

Name	Nb args	Arg. Type	Return Type
LT	2	1st: *InputValue*, 2nd *FeatValue*	*Bool*
IF	3	1st: *Bool*, 2nd and 3rd: *RegValue*	*RegValue*
EFV	0	-	*FeatValue*
ERV	0	-	*RegValue*

2.2. Logical consistency

Based on these ideas we show in Figure 2 two prototypical models that are type consistent. Both models perform two successive tests involving the *InputValue* primitive labeled `IN0`. The root of both trees, indicated by an arrow, first tests whether `IN0` is lower than a constant equal to 0. If the test succeeds, an ERV primitive connected to the second argument of the root is returned. Otherwise, a new test is performed via the right-most IF primitive, checking whether `IN0` is either lower than 1 (upper model), or -1

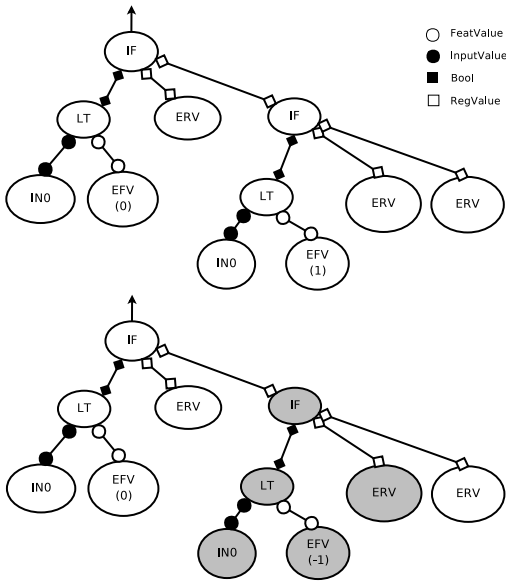

Figure 2. Two type-consistent regression trees. (Top) Logically consistent tree, (Bottom) Logically inconsistent tree

(lower model). We can see that the upper model is logically consistent: its tests are plausibly structured and can possibly represent the feature space. Oppositely, the lower model is logically inconsistent: because the right-most test is not logically consistent with the left-most one, all the primitives marked in gray turn to be useless. This leads to a situation of *code bloat* [11] where the evolved tree may contain useless branches, called *introns*. The influence of code bloat in the results of an evolutionary system is still debated and may depend of the model application domain. However, in this paper, because our work is preliminary, we decide to discard individuals containing such introns by introducing a logical consistency check mechanism. As a future extension, we plan to study thoroughly the influence of such introns in the evolutionary process for our particular domain.

2.3. Genetic Operators

In order to implement the logical consistency check presented above, we slightly modified the operators provided in [1], which are a refinement of the strongly typed operators proposed in [8]. We modified the base class of each of these operators to implement a logical consistency check. The main operators are Tree Crossover, where two individuals can swap subtree and Standard Mutation, where a subtree is replaced by a newly generated one. Additionally, Shrink mutation replaces a branch with one of its child nodes, and Swap mutation swaps two subtrees of an individual.

2.4. Constant Generation

Finally we use the ERV (respectively EFV) primitive for generating constants that are used for prediction (respectively comparison with inputs). ERV and EFV initialization operators are used to integrate the distribution of both inputs and outputs in the training data. We approximate these distributions with gaussians, and use the latter when generating random constants. This is a first step towards using a larger repertoire of statistical distributions (e.g. gamma distributions, mixture of gaussians).

We defined in this section a framework for evolving regression trees from the typing and logical point of view. Additionally, we use a mechanism for constant generation that reflects both input and output distributions of the training data. What we need now is a fitness measure in the context of performance modeling.

3. Accuracy for performance modeling: fitness evaluation

In this section, we devise a fitness function to guide the model search in the right direction. The fitness function performs a performance similarity measurement using the edit-distance. The formulas we introduce apply to the fitness computation for a given performance fragment. The average model fitness is computed over the distinct fragments of the training database.

3.1. Performance similarity based on edit-distance

How accurately can a given model learn to generate expressive fragments? We want to measure how similar is a predicted performance with the training one. To achieve this, we use a edit-distance that was fit to human performance similarity judgments, as shown below. The edit-distance [7] is used to determine the similarity between two performances of the same melody. It is defined as the minimal total cost of a sequence of editions needed to transform one performance into the other, given edit-operations like deletion, insertion, and replacement. The edit-distance is flexible enough to accommodate for comparison of performances of different lengths (in case of e.g. consolidation/fragmentation) and it allows for customization to a particular use by adjusting parameter values of the edit-operation cost functions. The cost of a particular edit-operation is defined through a cost function w that computes the cost of applying that operation to zero or more notes of one performance and zero or more of the other, given as parameters to w. We defined the following cost functions for 1-0, 0-1, 1-1, N-1, and 1-N mapping edit-operations, which we write respectively $w(\mathbf{s}_i, \emptyset)$, $w(\emptyset, \mathbf{t}_j)$, $w(\mathbf{s}_i, \mathbf{t}_j)$, $w(\mathbf{s}_{i-K:i}, \mathbf{t}_j)$ and $w(\mathbf{s}_i, \mathbf{t}_{j-L:j})$:

$$
\begin{aligned}
w(\mathbf{s}_i, \emptyset) &= \alpha_1 \left(\delta \mathcal{D}(s_i) + \epsilon \mathcal{E}(s_i) \right) + \beta_1 \\
w(\emptyset, \mathbf{t}_j) &= \alpha_1 \left(\delta \mathcal{D}(t_j) + \epsilon \mathcal{E}(t_j) \right) + \beta_1 \\
w(\mathbf{s}_i, \mathbf{t}_j) &= \alpha_2 \begin{pmatrix} \pi \left| \mathcal{P}(s_i) - \mathcal{P}(t_j) \right| + \\ \delta \left| \mathcal{D}(s_i) - \mathcal{D}(t_j) \right| + \\ o \left| \mathcal{O}(s_i) - \mathcal{O}(t_j) \right| + \\ \epsilon \left| \mathcal{E}(s_i) - \mathcal{E}(t_j) \right| \end{pmatrix} + \beta_2
\end{aligned}
$$

$$w(\mathbf{s}_{i-K:i}, \mathbf{t}_j) = \alpha_3 \begin{pmatrix} \pi \sum_{k=0}^{K} |\mathcal{P}(s_{i-k}) - \mathcal{P}(t_j)| + \\ \delta \, |\mathcal{D}(t_j) - \sum_{k=0}^{K} \mathcal{D}(s_{i-k})| + \\ o \, |\mathcal{O}(s_{i-K}) - \mathcal{O}(t_j)| + \\ \epsilon \sum_{k=0}^{K} |\mathcal{E}(s_{i-k}) - \mathcal{E}(t_j)| \end{pmatrix} + \beta_3$$

$$w(\mathbf{s}_i, \mathbf{t}_{j-L:j}) = \alpha_3 \begin{pmatrix} \pi \sum_{l=0}^{L} |\mathcal{P}(s_i) - \mathcal{P}(t_{j-l})| + \\ \delta \, |\mathcal{D}(s_i) - \sum_{l=0}^{L} \mathcal{D}(t_{j-l})| + \\ o \, |\mathcal{O}(s_i) - \mathcal{O}(t_{j-L})| + \\ \epsilon \sum_{l=0}^{L} |\mathcal{E}(s_i) - \mathcal{E}(t_{j-l})| \end{pmatrix} + \beta_3$$

where $\mathbf{s}$ and $\mathbf{t}$ are the performances as sequences of performed notes. Subsequences are denoted $\mathbf{s}_{i:j} = (s_i, ..., s_j)$, and $\mathbf{s}_i = (s_i)$, and $\mathcal{P}(s_i)$, $\mathcal{D}(s_i)$, $\mathcal{O}(s_i)$, and $\mathcal{E}(s_i)$ respectively represent the pitch, duration, onset, and dynamics attributes of a note s_i. Each attribute has a corresponding parameter (π, δ, o, and ϵ, respectively), that controls the impact of that attribute on operation costs. The β parameters control the absolute cost of the operations. The α parameters control the partial cost of the operation due to (differences in) attribute values of the notes.

3.1.1. Fitting the edit-distance to human judgments

The cost parameters presented above were optimized in [4] to fit human performance similarity judgments based on a web survey. In this survey, 92 subjects were presented questions containing target performance *A* (the nominal performance, without expressive deviations) of a short musical fragment, and two different performances *B* and *C* of the same fragment. The task was to indicate which of the two alternative performances was perceived as most similar to the target performance. The edit-distance cost parameters were optimized using a genetic algorithm to fit the survey results. We show in table 3 the final parameters values.

The optimized edit-distance is used to compute the fitness of the model, by comparing the model output performance to the training performance. We obtain the value of the error-driven fitness component by applying the following formula:

$$f_{error} = \frac{1}{1 + editdistance} \quad (1)$$

4. Experimental results and discussion

4.1. Data

In this paper we use an expressive performance database that comes from annotations of acoustic saxophone recordings. We consider four jazz from standards the *Real Book*. The excerpts are the following: *Body and Soul*, *Once I Loved*, *Up Jumped Spring*, *Like Someone In Love*. We characterize the score melodic context of this data using, for each note, the following features: note duration, metrical strength within the bar, previous and next note relative duration and previous and next note relative interval. We analyze the corresponding acoustic recordings to obtain the performance transcriptions. See [3] for detailed description of the transcription process.

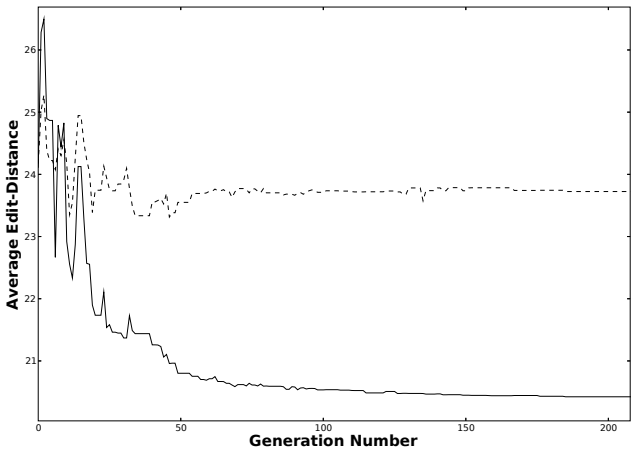

Figure 3. Average edit-distance of best-of-run model during the evolution. Plain: training data, dashed: test data

4.2. Method

The training and generalization experiments we present here are completed as follows: we include in the training data (used for calculating the fitness) the musical data that comes from the first two thirds of each of the four musical fragments presented above. The material of the last third of each fragment is used as test data. That is, we train the model to predict the performance events that take place during the first part of each fragment. During the test phase, we evaluate how accurately the model is able to predict the performance events from the end of these fragments. The test fragments were carefully chosen so that they do not contain melodic information present in the training set.

4.3. Evolutionary settings and Implementation

We used the following evolutionary settings for this experiment. The population size was set to 1000 individuals, and the maximum number of generations was set to 200. We used the following probabilities for our evolutionary operators: Crossover probability is set to 0.9, with a branch-crossover probability of 0.2. The Standard mutation probability is set to 0.01, while the Swap mutation has been set to 0.1. Finally, Shrink mutation probability is set 0.05. The maximum tree depth has been set to 20.

4.4. Results

Figure 3 shows a plot of the average edit distance over the training fragments (solid) and the test fragments (dashed) for the best-of-run model during the evolution. The x-axis indicates the generation number while the y-axis shows the average edit distance for training and test set. We can notice a first period in which the overall distance to the actually performed fragments is globally decreasing until generation 20. After this period, we can observe a clear tendency: the training distance decreases while the test distance does not

Table 3. Optimized values of edit-operation cost parameters

α_1	α_2	α_3	β_1	β_2	β_3	π	δ	o	ϵ
0.031	0.875	0.243	0.040	0.330	0.380	0.452	1.000	0.120	0.545

decrease anymore. This is a situation of overfitting, which might be a consequence of the limited amount of training data. To confirm this, we can see that the maximum depth of the evolved models is 11, while the tree programs were allowed to grow to a depth of 20. This means that the training data are not sufficient for the models to grow enough and find a system of rules that is suitable for both training and test set.

5. Conclusion

We have presented an approach to derive evolutionary regression trees for modeling expressive music performance. We first presented the benefits of evolutionary computation and strongly typed genetic programming, which can be summarized as follows: evolution of a population of models, prior knowledge integration, and flexibility regarding the data types, structures, and model evaluation. In continuation, we defined the basis of a structured-prediction regression tree for modeling expressive music performance. We specified the types, primitives, operators and fitness function used in the STGP framework. We have set the basis of a performance modeling evolutionary framework based on a performance similarity measure that was fit to human similarity judgments. We finally presented preliminary results for both learning and generalization experiments, using a small expressive performance database annotated from monophonic recordings of jazz standards. We compared the training and test performance similarities, to identify an overfitting point.

We plan to extend our work in the following directions: first we want to increase substantially the size of the annotated performance database we are using in order to more robustly assess the training and generalization abilities of the evolved models. We will take advantage of recent works such as [2] that highlight the importance of a validation set for selecting robust best-of-run models. At this point, it is worth mentioning that the incorporation of an edit-distance based fitness computation into the evolutionary framework has led to a massive use of computation time: increasing substantially the training database will require us to set a distributed framework, which is featured in Open Beagle. Also, we will integrate an extended set of performance events in order to reflect all the possible manifestations of expressivity that characterize jazz performance. Finally, in our actual approach we achieve score-driven performance prediction. In the context of sequential processing, an exciting work direction is studying the contribution of both score context and past expressive gestures in music performance.

References

[1] C. Gagné and M. Parizeau. Open beagle manual. Technical Report RT-LVSN-2003-01-V300-R1, Laboratoire de Vision et Systèmes numérique, Université de Laval, 2005.

[2] C. Gagné, M. Schoenauer, M. Parizeau, and M. Tomassini. Genetic programming, validation sets, and parsimony pressure. In P.Collet et al., editor, *EuroGP 2006, LNCS 3905*, pages 109–120, 2006.

[3] E. Gómez, M. Grachten, X. Amatriain, and J. Arcos. Melodic characterization of monophonic recordings for expressive tempo transformations. In *Proceedings of Stockholm Music Acoustics Conference 2003*, Stockholm, Sweden, 2003.

[4] M. Grachten, J.L. Arcos, and R. López de Mantaras. A case based approach to expressivity-aware tempo transformation. *Machine Learning (in press)*, 2006.

[5] A. Hazan, R. Ramirez, E. Maestre, A. Perez, and A. Pertusa. Modelling expressive music performance: a regression tree approach based on strongly-typed genetic programming. In *Forth European Workshop on Evolutionary Music and Art*, pages 676–687, Budapest, Hungary, 2006.

[6] J.R. Koza. *Genetic Programming: On the programming of Computers by means of natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[7] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[8] D.J. Montana. Strongly typed genetic programming. Technical Report 7866, 10 Moulton Street, Cambridge, MA 02138, USA, 7 1993.

[9] R. Ramirez and A. Hazan. A tool for generating and explaining expressive music performances of monophonic jazz melodies. In *International Journal on Artificial Intelligence Tools (to appear)*, 2006.

[10] R. Ramirez, A. Hazan, and E. Maestre. Intra-note features prediction model for jazz saxophone performance. In *International Computer Music Conference*, Barcelona, Spain, 2005.

[11] Walter Alden Tackett. Greedy recombination and genetic search on the space of computer programs. In *Foundations of Genetic Algorithms 3*. Morgan Kaufmann, San Francisco, CA, 1995.

[12] A. Tobudic and G. Widmer. Relational IBL in music with a new structural similarity measure. In *Proceedings of the International Conference on Inductive Logic Programming*, pages 365–382, 2003.

[13] G. Widmer. In search of the horowitz factor: Interim report on a musical discovery project. In *Proceedings of the 5th International Conference on Discovery Science (DS'02)*, Lbeck, Germany., 2002.

[14] G. Widmer. Machine discoveries: A few simple, robust local expression principles. *Journal of New Music Research*, 31(1):37–50, 2002.