Modelling Expressive Performance: A Regression Tree Approach Based on Strongly Typed Genetic Programming

Amaury Hazan[1], Rafael Ramirez[1], Esteban Maestre[1],
Alfonso Perez[1], and Antonio Pertusa[2]

[1] Music Technology Group, Pompeu Fabra University,
Ocata 1, Barcelona 08003, Spain
ahazan@iua.upf.es, rafael@iua.upf.es, emaestre@iua.upf.es,
aperez@iua.upf.es
[2] Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante,
Alicante, Spain
pertusa@dlsi.ua.es

Abstract. This paper presents a novel Strongly-Typed Genetic Programming approach for building Regression Trees in order to model expressive music performance. The approach consists of inducing a Regression Tree model from training data (monophonic recordings of Jazz standards) for transforming an inexpressive melody into an expressive one. The work presented in this paper is an extension of [1], where we induced general expressive performance rules explaining part of the training examples. Here, the emphasis is on inducing a *generative* model (i.e. a model capable of generating expressive performances) which covers all the training examples. We present our evolutionary approach for a one-dimensional regression task: the performed note duration ratio prediction. We then show the encouraging results of experiments with Jazz musical material, and sketch the milestones which will enable the system to generate expressive music performance in a broader sense.

1 Background

1.1 The Expressive Performance Modelling Problem

Modelling expressive music performance is one of the most challenging aspects of computer music. The focus of this work is to study how skilled musicians (here a Jazz saxophone player) express and communicate their view of the musical and emotional content of musical pieces by introducing deviations and changes of various parameters. The expressive performance modelling problem can be stated as follows. We define an expressive performance database B that consists of a set of pairs (S_i, E_i), where S_i is a set of melodic features of a given score note N_i, and E_i is a set of acoustic features describing the expressive transformations applied to note N_i. The problem is to find a model M that will minimise the

F. Rothlauf et al. (Eds.): EvoWorkshops 2006, LNCS 3907, pp. 676–687, 2006.

error $Err(M(S_i)$, Ei) between the prediction $M(S_i)$ and the actual expressive transformation E_i for all the pairs (S_i, E_i) in B. Typically S_i is a set of features describing the melodic context of note N_i, such as its melodic and rhythmic intervals with its neighbours or its metrical position in the bar. On the other hand, E_i contain local timing and overall energy information of the performed note, but can also contain finer-grain information about some intra-note features (e.g. energy envelope shape, as studied in [2], or pitch envelope).

1.2 Regression Trees for Expressive Performance Modelling

In the past, we have studied expressive deviations on note duration, note onset and note energy [3]. We have used this study as the basis of an inductive content-based transformation system for performing expressive transformation on musical phrases. Among the generative models we induced, Regression Trees and Model Trees (an extension of the former) showed the best accuracy.

Regression Trees, are widely used in pattern recognition tasks, each non-leaf node in a Regression Tree performs a test on a particular input value (e.g. inequality with a constant), while each non-leaf node is a number representing the predicted value. By using a succession of IF-THEN-ELSE rules, Regression Trees iteratively split the set of training examples into subsets where the prediction can be achieved with increasing accuracy. The resulting structure is a coherent set of mutually-excluding rules which represents the training set in a hierarchical way. Figure 1 shows an example of a simple Regression Tree. This tree performs tests on 2 different features and returns a numerical prediction at its leaves. Whether this tree can produce accurate predictions when processing unseen data depends directly on the generalisation power of the model. To ensure good generalisation capability, tree induction algorithms, such as C4.5 [4] and M5 [5], provide pruning operations that rely on statistical analysis on the set of training examples.

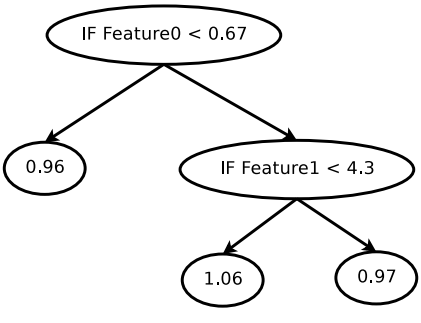

Fig. 1. A simple Regression Tree example. Tests are performed on 2 different features, namely Feature0 and Feature1. If a test succeeds, the left-side children is executed, while if the test fails, the right-side children is executed. When reaching a leaf, a numerical prediction is returned.

1.3 Evaluation Methodology for Expressive Transformation Data

We would like to add in the building process an evaluation of the model when predicting expressive transformations of musical excerpts in different musical situations. An example is to measure the system's ability to predict transformations of a given musical fragment at a given tempo when the model was built using training data based on recordings of the same tune but played slightly faster. Another example would be to compare the system behaviour when providing predictions on two dissimilar tunes when it was trained using data representing only one of these. In this work, we intend to evaluate the ability of our model to solve expressive musical performance situations considering these aspects.

1.4 An Evolutionary Computation Perspective

Most of tree induction algorithms are *greedy*, e.g. trees are induced top-down, a node at a time. According to [6], several works pointed out the inadequacy of greedy induction for difficult concepts. An alternative is to use Evolutionary Computation techniques, and especially Genetic Programming to evolve Regression Trees. By combining the high-level representation of computer programs and the near-optimal efficiency of learning with the parallel processing of several potential solutions, the Genetic Programming framework is an interesting direction for building such models. Koza [7] shows that this method has been successfully applied to concept learning and empirical discovery tasks.

Background on Genetic Programming
Genetic Algorithms (GA) can be seen as a general optimisation method that searches a large space of candidate hypotheses seeking one that perform best according to a fitness function. As presented in [8], GA transform a population of individuals, each with an associated value of fitness (i.e. ability to solve the problem), into a new generation of the population. The new generation is obtained using the Darwinian principles of survival and reproduction of the fittest and analogs of naturally occurring genetic operations such as crossover and mutation. In [7], Koza presents the Genetic Programming (GP) extension. While in the GA framework individuals are typically represented as binary or float strings, in the GP paradigm the structures undergoing adaptation are hierarchical computer programs of dynamically varying size and structure. These computer programs are represented as trees, in which a node represents a function, while each leaf is a terminal (e.g. input of the computer program).

Strongly Typed Genetic Programming
According to Montana [9], in standard GP, there is no way to restrict the programs it generates to those where the functions operate on appropriate data types. When the programs manipulate multiple data types and contain functions designed to operate on particular data types, this can lead to unnecessarily large search times and/or unnecessarily poor generalisation performance. This analysis is crucial in the context of building a Regression Tree model for expressive music performance. Indeed, tests operate on input variables

of the program, that is, inputs are compared to inputs which appear in the training set. On the other hand, leaves contain predictions which should reflect the distribution of outputs E_i in the training set. Although both inputs and predictions of our model can be coded as float values, building a program that performs direct comparisons between an input value, and an output value (i.e. a melodic feature in S_i and an expressive feature in E_i), would produce ineffective computations, and a considerable effort would be spent in order to generate individuals that reasonably fit the training examples. Montana presented an enhanced version of GP called Strongly Typed Genetic Programming (STGP) which enforces data type constraints. Using this approach, it is possible to specify what type of data a given GP primitive can accept as argument (e.g. type of input), or can return (e.g. output). To the best of our knowledge, this work presents a novel approach for building Regression Trees using STGP.

In section 2, we describe how to represent the problem and implement the structural constraints of Regression Trees using STGP types. Preliminary results in predicting one-dimensional expressive transformations of musical material are then presented in 3. Finally, Section 4 draws some conclusions and future work.

2 An Approach for Building a STGP Regression Tree Music Performance Model

2.1 Performance Training Data

Each score note N_i in the performance database B is annotated with a number of attributes representing both properties of the note itself and some aspects of the local context in which the note appears. Information about intrinsic properties of the note include the note duration and the note metrical position, while information about its context include the duration of the previous and following notes, and the extension and direction of the intervals between the note and the previous and following notes. Thus, each S_i in B contains six features sumarising this melodic information. In the last section, we will discuss the perspective of generating new melodic representations for building expressive models. Each note N_i is associated to a set of acoustic features E_i describing how the note was played by the performer. This information was obtained by applying a segmentation algorithm on the audio recordings and then performing an alignment of the annotated audio with the score (see [10] for a detailed description of these two steps). In this work, we focus on the duration ratio of the performed note, i.e. ratio between the performed note duration and the score note duration. This is obviously a first step before considering an extended set of expressive features as presented in [2] and [3]. Because this work is preliminary, we limited our training set to a few examples of the expressive performance database we maintain, namely two versions of the excerpt *Body And Soul*, performed at 60 and 65 beats per minute, and two versions of *Like Someone In Love*, performed at 120 and 140 beats per minute.

2.2 Types and Primitives

Going back to Figure 1 we can see that the general structure of the Regression Tree is the following. Each node is a test comparing an input with a value that was generated analysing the training data. Each leaf is an output value containing the numerical prediction. Thus, both inputs and outputs of the program should have different types. We define 4 different types, namely *InputValue*, *FeatValue*, *RegValue* and *Bool*. The first three types represent floating-point values, while the latter type represents boolean values that will be used when performing tests. Now we can define the primitives that will be used to build our models. They are listed in Table. 1.

Table 1. STGP primitives used in this study. Each primitive is presented along with the number of arguments it handles, the type of each argument, and its return type. Note that primitives EFeatValue and ERegValue, which are used for constant generation, take no argument as input.

Primitive Name	Number of arguments	Arguments Type	Return Type
IF	3	1st: *Bool*, 2nd and 3rd: *RegValue*	*RegValue*
LT	2	1st: *InputValue*, 2nd *FeatValue*	*Bool*
EFeatValue	0	-	*FeatValue*
ERegValue	0	-	*RegValue*

The **IF** primitive tests whether its first argument of type *Bool* is true or false. If the test succeeds, its second argument is returned, otherwise its third argument is returned (both second and third arguments have a *RegValue* type). The **LT** primitive tests whether its first argument is lower than the second argument (The former has a *InputValue* type and the latter *FeatValue*). The primitive returns a *Bool* which value is true if the test succeeds, false otherwise. Given the definitions of our types, we can see that during the building process the output of the **LT** primitive will always be connected to the first argument of the **IF** primitive. Instead, we could have chosen to use a single primitive **IFLT** with 4 arguments (the first two being involved in *InputValue* typed comparison, and the last two being used to return a *RegValue*). However, we think that restricting the tests to inequalities would be a constraint for future developments. *EFeatValue* and *ERegValue* are zero-argument primitives. We use them for generating constants. The first one generates constants to be involved in inequality tests (primitive **LT**). In order to produce meaningful tests, values produced by *EInput* are likely to be the values appearing in the vectors S_i of the training set. To ensure this, we collected all the features in S_i in B. When creating a *EFeatValue* primitive, we choose randomly one of them. Similarly, *ERegValue* primitive generates constants of type *RegValue* that will form the numerical predictions returned by the model. It is desirable that these predictions respect

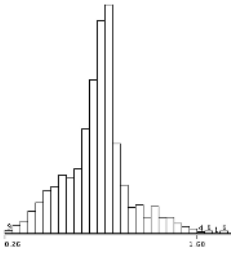

Fig. 2. Duration ratio distribution in the training data

approximately the statistical distribution of the vectors E_i in B. In this case, we focus on the single duration ratio in E_i. After having analysed the distribution of this value (see Figure 2.2), we decide to randomly generate *ERegValue* following a gaussian distribution fitting the training data.

2.3 Terminal Set

The elements of the terminal set are the inputs of the program. Each of the features of test vector S_i is associated to an input of the tree $IN_i, 0 \leq i \leq 5$. Terminals are of type *InputValue*, thus they will only feed primitives accepting arguments of this type (i.e. the LT primitive in the present case). Given the types, primitives, and terminals presented above, we are now able to build a Regression Tree structure in the STGP framework. Figure 3 shows how the example tree introduced in Figure 1 is represented.

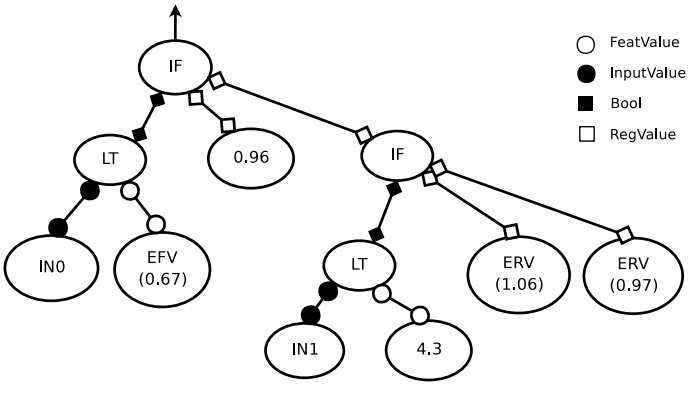

Fig. 3. STGP Regression Tree example involving two successive tests. Typed connections between primitives are appearing. Also, constants generated by zero-arguments primitives appear in parentheses. EFV stands for EFeatValue, while ERV stands for ERegValue. The arrow indicates the output point of the program.

2.4 Genetic Operators

The genetic operators we use are basically a refinement of the operators proposed by [7], and [9] in their Strongly Typed variant, where the operation is constrained to produce a type-consistent structure. Operators are Tree Crossover, where two individuals can swap subtrees, 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. Also, two floating-point mutation operators where added to process the randomly-generated constant returned by *EInput* and *ERegValue* primitives.

2.5 Fitness Evaluation

Individuals are evaluated on their ability to produce an output $M(S_i)$ that matches E_i for each note N_i of a musical fragment in the training set. Consequently, the fitness measure is the average error of the model $Err(M(S_i)$, Ei). We use the following fitness function:

$$f = \frac{1}{1 + RMSE} \tag{1}$$

where RMSE is the Root Mean Squared Error between training and predicted duration ratio, averaged over the notes of a musical fragment, itself averaged over all the training fragments. We are aware that this fitness measure is very general and that it does not catch in detail the behaviour of the model (e.g. evolution of the error note by note during the performance of a musical fragment). We have no guarantee that the best ranked individual will be the model that performs the best from the musical point of view. However this measure gives a first search direction during the evolution. Future developments will take advantage of perceptually-based fitness measures, but prior to this we have to address the issue of multi-dimensional prediction (see Sec. 4) in order to generate new melodies.

2.6 Evolutionary Settings and Implementation

We define the following evolutionary settings for the different runs. The population size is fixed to 200 individuals. The evolution stops if 500 generations have been processed or if the fitness reaches a value of 0.95. We use a generational replacement algorithm with a tournament selection. Crossover probability is set to 0.9, with a branch-crossover probability of 0.2, which means that crossovers are more likely to be performed on leaves, with the effect of redistributing the constants and terminals in the tree. The Standard mutation probability is set to 0.01, while the Swap mutation has been set to a higher value (0.1) in order to let a individual reorganise its feature space partition with more probability. Finally, Shrink mutation probability is set 0.05. The maximum tree depth has been set to 10, which could lead to the generation of very complex individuals (here we do not look for a compact model).

We build our system using Open Beagle framework [11], which is a C++ object oriented framework for Evolutionary Computation. Open Beagle was primarily designed for solving Genetic Programming tasks and includes STGP features such as the operators presented above.

3 Results and Evaluation in Different Musical Situations

In this section, we test the ability of our model in predicting expressive transformations given different training situations. Note that in this section we will not focus on the fitness of the individuals as defined in last section. Rather, we evaluate a model based on its error prediction. First, we address the problem of evaluating precision and generalisation capability of the induced model in the context of expressive music performance. In Figure. 4, we present the RMSE of the best-so-far model when predicting the note duration ratio of the four excerpts presented above. On the top, the model is only trained with one of the four fragments, namely *Body and Soul* played at 60 beats per minute, i.e, the fitness function is only based on this error. On the bottom, the model is trained using the four fragments. First we can see that excerpts that share the same

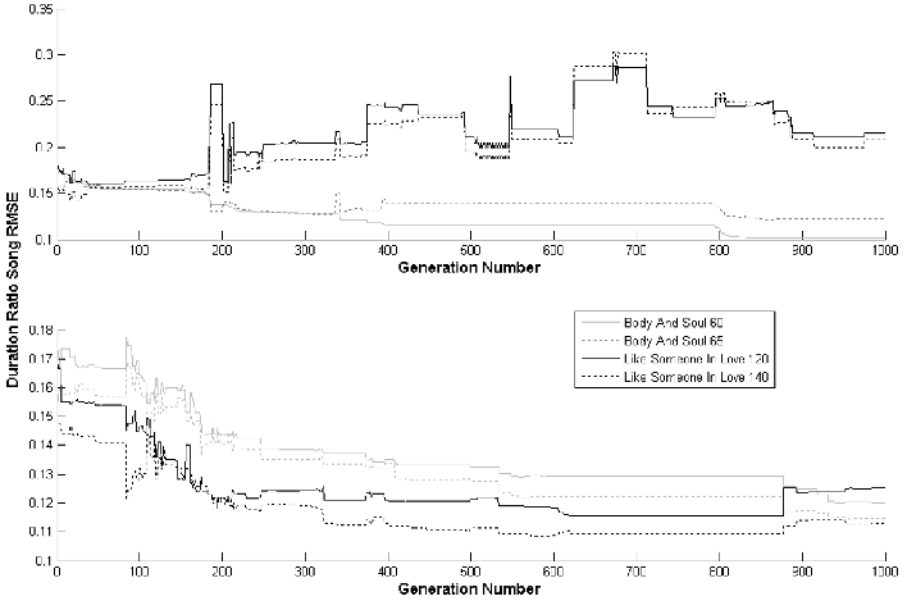

Fig. 4. Best-so-far individual RMSE for each of the target songs when the fitness takes into account only the *Body And Soul* (played at 60 beats per minute) prediction error (top), or when the fitness takes into account prediction the error of the four fragments (bottom). Gray solid (respectively dashed) line is the RMSE of *Body And Soul* played at 60 (respectively 65) beats per minute. Black solid (respectively dashed) line is the RMSE of *Like Someone In Love* played at 120 (respectively 140) beats per minute.

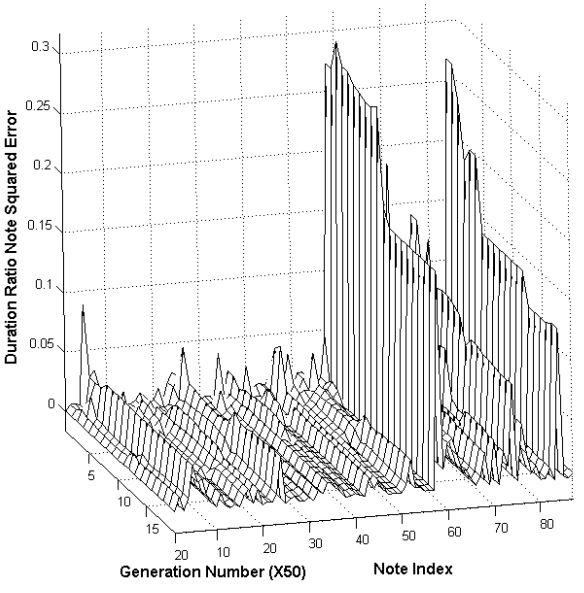

Fig. 5. Note by note Squared Error of the duration ratio during the evolution process

melodic representation evolve in a similar fashion. In the case of being trained with only one song (top of Figure. 4), the prediction error start to differentiate at generation 200. After this point, predictions concerning the training and similar fragments become better while predictions concerning dissimilar fragments become worse. On the other hand (bottom of Figure. 4), when all the fragments are used during training, the error tends to be minimised for the four excerpts. Some modifications of the model improve the prediction for all excerpts (e.g. generation 530), while others (e.g. generation 870) benefit only to excerpts sharing the same melodic features.

Figure. 5 shows the note by note Squared Error of the duration ratio during the evolution process. First generation model error appear on the back while last generation model error is on the front. We can see, that even if the overall error is minimised, some notes, such as note 61, 80, and 85 are modelled with less accuracy. This means that the best tree model does not cover these melodic situations appropriately. New specific branches should be created to perform tests on melodic features that represent theses melodic situations.

Finally, in Figure. 6, we perform an informal comparison between two models and the training data when predicting the duration ratio of each note the excerpt *Like Someone In Love* played at 140 beats per minute. The first model is the best-so-far STGP model obtained for this task (indicated in grey dotted line). The second model (grey dashed line) was obtained using a propositional greedy Regression Tree algorithm (see [3] for details), which is an accurate technique for the performance modelling task. The black solid line corresponds to the actual training data for this excerpt. We can see that both STGP and greedy Regression

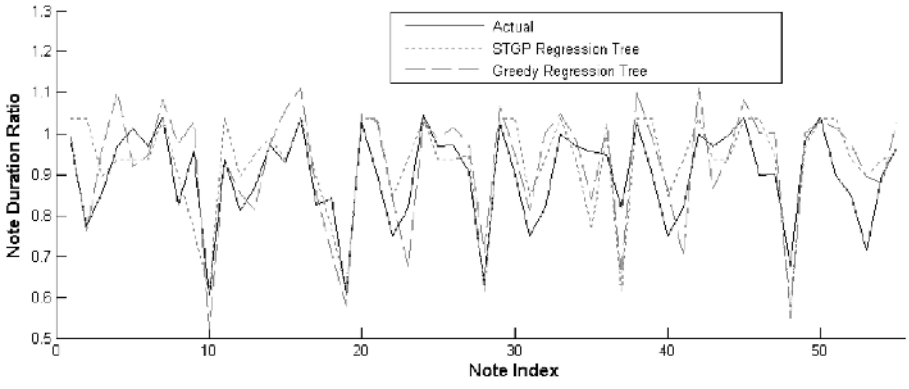

Fig. 6. Note by note duration ratio prediction for *Like Someone in Love* played at 140 beats per minute. Black solid line refers to the performance training data, grey dotted line refers to the best-so-far STGP model, and grey dashed line corresponds to a greedy Regression Tree model as presented in [3].

Tree models behave qualitatively well, and their mean prediction error is very similar. Thus, our approach to build Regression Tree performance models based on STGP is promising, although the results we present here are very preliminary.

4 Conclusion

We presented in this work a novel Strongly Typed Genetic Programming based approach for building Regression Trees in order to model expressive music performance. The Strongly Typed Genetic Programming framework has been introduced, along with the primitives, operators and settings that we apply to this particular task. Preliminary results show this technique to be competitive with greedy Regression Tree techniques for a one-dimensional regression problem: the prediction of the performed note duration ratio. We want to scale up our approach to the generation of expressive musical performances in a broader sense, and plan to work in the following directions:

– Predict more expressive features, such as onset and mean note energy deviation, which will enable the model to predict expressive melodies. This will be achieved by defining a new *RegValue* complex data type, along with an appropriate constant generator and operators. New fitness measures have to be defined and in order to assert the musical similarity between the model's output and the performer's transformations. We believe that the use of perceptually motivated fitness measures (e.g. [12]) instead of statistical errors (e.g. RMSE) can lead to substantial improvements of the accuracy of our models and make the difference with classical greedy techniques. Additionally, intra-note features are of particular interest in monophonic performances and will be considered. This will include energy, pitch, and timbrical features.

- Generate new melodic representations for our expressive performance database: as pointed out in Section. 2, the melodic representation we use is based on musical common sense but is not necessarily the best one. A particular drawback of this representation is that it only capture local context information, when several works stress that expressive music performance is a complex phenomenon which involves a multi-level representation of the music. An interesting work perspective is to devise a evolutionary melodic feature extractor that would coevolve with the performance model. [13] presents relevant ideas to achieve this.
- Increase the amount of training data. We are aware that our database is still small and that much more data is needed. New methods for the acquisition of performance data from polyphonic recordings (which would allow us to obtain data from commercial recordings) start to give interesting results. It is mandatory to build a representative performance database if we want to investigate thoroughly the generalisation (from both statistical and musical point of of view) power of the induced models. We will use statistical tests (e.g. 10-fold or one-song-out cross validation) to assess the performance of a given model. However we want to keep track of the sequences of events we are modelling, consequently we will avoid to use any technique which would lead to loose the temporal continuity in the data.
- Towards a model with state. A drawback in the model architecture is that it only bases its predictions on the melodic features of the note to be transformed, while it should also have access to the past predictions it returned. This last aspect is an important issue of the challenging expressive music performance problem.

References

1. Ramirez, R., Hazan, A.: Understanding expressive music performance using genetic algorithms. In: Third European Workshop on Evolutionary Music and Art, Lauzane, Switzerland (2005)
2. Ramirez, R., Hazan, A., Maestre, E.: Intra-note features prediction model for jazz saxophone performance. In: International Computer Music Conference, Barcelona, Spain (2005)
3. Ramirez, R., Hazan, A.: A tool for generating and explaining expressive music performances of monophonic jazz melodies. In: International Journal on Artificial Intelligence Tools (to appear). (2006)
4. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc. (1993)
5. Quinlan, J.R.: Learning with Continuous Classes. In: 5th Australian Joint Conference on Artificial Intelligence. (1992) 343–348
6. Murthy, S.: Automatic construction of decision trees from data: A multi-disciplinary survey. Data Mining and Knowledge Discovery **2** (1998) 345–389
7. Koza, J.: Genetic Programming: On the programming of Computers by means of natural Selection. MIT Press, Cambridge, MA, USA (1992)
8. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbour, MI (1975)

9. Montana, D.: Strongly typed genetic programming. Technical Report #7866, 10 Moulton Street, Cambridge, MA 02138, USA (1993)
10. Gómez, E., Grachten, M., Amatriain, X., Arcos, J.: Melodic characterization of monophonic recordings for expressive tempo transformations. In: Proceedings of Stockholm Music Acoustics Conference 2003, Stockholm, Sweden (2003)
11. Gagné, C., Parizeau, M.: Open beagle manual. technical report rt-lvsn-2003-01-v300-r1. Technical report, Laboratoire de Vision et Systèmes numérique, Université de Laval (2005)
12. Grachten, M., Arcos, J., López de Mántaras, R.: Melody retrieval using the implication/realization model. In: Online Proceedings of the 6th International Conference on Music Information Retrieval, London, UK (2005)
13. Conklin, D., Anagnostopoulou, C.: Representation and discovery of multiple viewpoint patterns. In: Proceedings of the International Computer Music Conference, La Havana, Cuba (2001) 479–485