Modelling expressive performance using consistent evolutionary regression trees

Amaury Hazan and **Rafael Ramirez** [1]

Abstract. We present an evolutionary approach for building regression tree based models in the context of expressive music performance. We first review the benefits of using evolutionary computation techniques in this context. We then use the strongly-typed genetic programming framework and define the types and constraints that are needed for evolving efficiently multi-dimensional regression trees, and present two fitness functions for modelling expressive performance local timing. While the first fitness measurement is purely error-driven, the second also takes into account the balance of the evolved tree in terms of input space representation. Finally, we present the results of both learning and generalization experiments. For these experiments, we use a database of saxophone performance timing extracted from a set of acoustic recordings of jazz standards. The whole system is built into the Open Beagle [5] evolutionary computation framework.

1 Introduction

Modelling expressive music performance is a challenging aspect for several research areas ranging from computer music to artificial intelligence and behavioral psychology. The aim of this paper is to study 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. In this paper we focus on the influence of the melodic context on the expressive gestures. The problem can be stated as follows: how do underlying patterns in the score account for expressive transformations in a performance? Our approach is the following: we extract a set of features describing the melodic context of a given set of scores, 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, the corresponding pieces performed by a human player are characterized with a set of expressive features such as timing, loudness or microtonal information. The modelling task lies in finding an appropriate mapping from melodic descriptors to expressive features that 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.

Evolutionary computation (EC) has been considered with growing interest in musical applications. Since [9], it has often been used in a compositional perspective, either to generate melodies ([4]) or rhythms ([15]). In [12] the harmonization subtask of composition is addressed, and a comparison between a rule-based system and a genetic algorithm is presented. EC has also been considered in improvisation applications such as [1], where a genetic algorithm-based model of a novice jazz musician learning to improvise is developed. On the other hand, despite the challenge it represents for pluridisciplinary research, inductive expressive performance modelling has motivated relatively few works. In [17, 16] the authors investigate loudness and tempo variations of classical piano performances and induce multi-level performance models. [7] presents a case based reasoning system to model expressivity-aware tempo transformations and makes use of genetic algorithms to optimize the melodic edit-distance involved in similarity computation.

This paper is an extension of [8], where we present the bases of evolutionary regression trees. Here, we aim to use evolutionary computation as the central framework for evolving music performance models, in particular regression trees, because they provide fined-grained multi dimensional predictions. Such predictions are used to generate expressive performance transformations on new musical material. In this paper, we focus on models *generating* with precision local note timing (namely note duration and onset deviation) and loudness prediction.

1.1 Benefits of using evolutionary computation techniques

We present in this section the motivations that led us to consider evolutionary computation, and more specifically strongly-typed genetic programming (STGP), 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 [10], we focus our attention on how these techniques can benefit our particular problem.

1.1.1 Population of transformation models

Human Performance is an inexact phenomenon: a musician could hardly reproduce identically a given performance. This is particularly true in jazz music, where the ”improvisational contribution” to a performance is fundamental, whereas this kind of variability is not accepted in classical music. 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 techniques provide an elegant way to cope with this limitation, with benefits for both end-users and researchers. First, the result of an evolution is a population of transformation models

[1] Pompeu Fabra University, Barcelona, Spain, email: {ahazan,rramirez}@iua.upf.es

that will eventually produce different musical transformations. Moreover, each model is *ranked* according to its fitness. The end-user can merely choose to keep the best-ranked model transformation. Alternatively he has the opportunity to navigate through the whole population and select the transformation of other models on an aesthetic basis. Conversely, by analyzing the whole population of models, and especially the best and worst-ranked models, the researcher has to opportunity to appreciate on several examples the results of specific design decisions and to refine the fitness function.

1.1.2 Integrating domain-specific knowledge

The use of EC techniques provides a way of integrating in the system elements of domain-specific knowledge. In the case of feature-based tree modelling, one can define this knowledge at two different levels, namely the tree structure, and the prior distribution of both model features and outputs. We detail the former in Section 2. Concerning the latter, 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, and these constants have to respect the statistical distribution of the feature space. On the other hand, single or even structured constant predictions (see next section) are generated independently to where they will appear in the tree model. Consequently, the role of the initialization operator is to provide a ”good guess” for both features and prediction values.

1.1.3 Custom data types for structured modelling and prediction

Although it is possible to use evolutionary computation in a straightforward way, i.e. using templates and well-defined data types, the EC programmer has the freedom to 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. In the present work, we produce multi-dimensional predictions by modifying the prediction data type of the genetic program, that is, the data type returned by the root primitive. Our approach is still scalable because STGP allows to define arbitrary data types (e.g. vectors, lists and compound structures) which could represent a set of expressive transformations. As a comparison, traditional greedy techniques derived from C4.5 [13] mainly produce unidimensional prediction models even if some more recent algorithms [2] feature multi-dimensional prediction. With these techniques, building models for predicting structured data would involve the use of separate and possibly inconsistent tree models.

We presented above several points for which we consider promising the use of evolutionary computation techniques for performance modelling. We acknowledge that other techniques implement part of these aspects. For instance, prior distributions are an important component in the Bayesian theory, while Random Forest techniques allows to produce a population of models. Nevertheless, to our knowledge, no other machine learning technique integrate these aspects in such a unified way.

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 necessary to produce consistent regression tree models from both typing and logical point of view. Section 3 introduces considerations for representing the input space as well as a suitable multi-dimensional error measurement. This leads us to define a pure transformation error-based fitness and another fitness that considers both input space representation and transformation error. In Section 4 we compare both learning and generalization accuracy of these two fitness measurements using an expressive performance database and we discuss the results. Finally, Section 6 presents our conclusions as well as future extensions to this work.

2 Consistent 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 (e.g. a *less than* operator). Each leaf represents a prediction. Our aim is to integrate these specificities 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 Type Consistency

We define 4 different types, namely *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. In this paper the prediction is encoded as a triplet of float values, however we are free to use any arbitrary data structure. We then define the corresponding set of regression tree primitives, as presented below.

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*

Table 1. STGP primitives used 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.

We now summarize the primitives presented in Table 1 and show how they correspond to the structure of a regression tree. First, LT primitive tests whether an input of type *InputValue* is less 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. This 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.

2.2 Logical Consistency

Based on these ideas we show in Figure 1 two prototypical models that are type consistent. Both models perform 2 successive tests

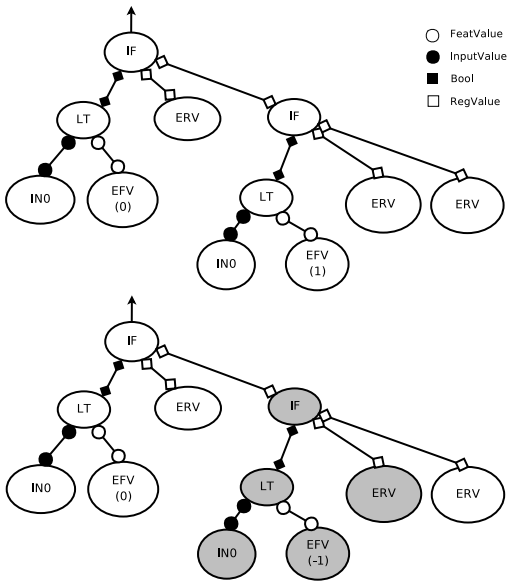

Figure 1. Two type-consistent regression trees. (Top) Logically consistent tree, (Bottom) Logically inconsistent tree

involving the *InputValue* primitive labelled `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 (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* [14] 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 decided 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 [5], which are a refinement of the strongly typed operators proposed in [11]. We modified the base class of each of these operators to implement a logical consistency check. For this, we take advantage of the prefix order indexing used in Open Beagle framework. Indeed, if a logically consistent tree (and any of its subtrees) built with the primitives presented in 1 is searched in prefix ordering (PO), the EFV primitives associated to a particular input appear sorted. We present the logical consistency check pseudo-code below:

```
INPUT: Tree S
OUTPUT: Boolean
REPEAT for i=1..n-inputs
  Collect IN_i indexes in S into L1
  WHILE L1 not empty
    REPEATfor each subtree s with IN_i in s:
      Collect in PO EFV values..
      ..compared to IN_i
      if (L2 not sorted): return false
      else: remove indexes referencing..
      ..IN_i in s from L1
    END
  END
END
return true
```

As a consequence, we obtained a modified version of the following operators: 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.

Constant generation: ERV and EFV initialization operators are used to integrate the distribution of the both inputs and outputs in the training data. The distribution of the three expressive features we want to predict is shown in Figure 2. We approximate the expressive transformation features distribution with gaussians, and use these distributions when generating random constants (with the ERV primitive). This is a first step towards using a larger repertoire of possible statistical distributions. For example, the relative energy variation distribution is rather asymmetric, and could be better modelled with a Gamma distribution.

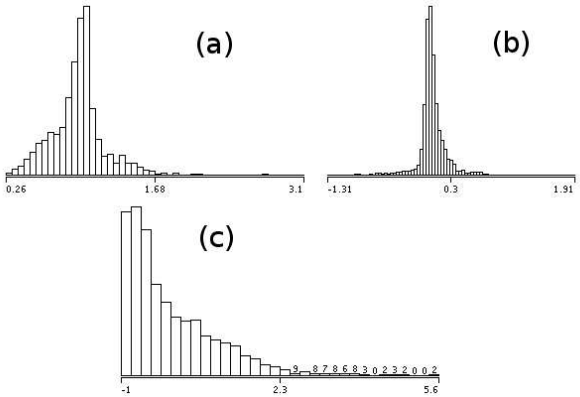

Figure 2. Distribution of expressive transformation features. (a) Duration ratio, (b) Onset deviation, (c) Energy Relative Variation

Concerning the constants used for comparison with the model inputs (EFV primitives), we use a somewhat simpler scheme: we collect the set of possible values for each input in the training set. During the EFV constant generation, we select one of the possible values of the corresponding input using a random uniform distribution. We defined in this section a framework for evolving regression trees that takes into account both typing and logical consistency constraints. Additionally, we use a mechanism for constant generation that reflects both input and output distribution of the training data. What we need now is a fitness measure in the context of performance modelling.

3 Model accuracy for performance modelling: Fitness evaluation

In this section, we investigate how to devise a fitness function able to guide the model search. We first introduce an unsupervised component promoting a balanced input space representation. Furthermore, we present an error-driven component suitable for multi-dimensional regression. In Section 4, we will compare a fitness function that takes both components into account with a pure error-driven measurement. Note that the formulas we introduce below apply to the fitness computation a given musical fragment. We then compute an average over the distinct fragments of the training database.

3.1 Balanced representation of the input space

We first describe a mechanism for promoting an accurate and balanced representation of the input space in the context of multi-dimensional regression. As pointed out in [6], pure error-driven fitness measurements in GP may lead to overfitting. Several papers have introduced the notion of *parsimony pressure* as a way to reduce the complexity of the models while improving the generalization performances. This is relevant in the case of e.g. binary classification problems, however in some situations (we believe our application domain is one of these) a bias toward low-complexity does not improve the generalization performance of the model ([3]). In our particular case, where the expressive transformations of musical excerpts result in delicate modifications in the timing, the simplest models would essentially let the score timing unaffected. Here, what we are interested in is a model that does not collapse rather dissimilar input vectors into the same leaf. In our case, this would mean that notes in very different contexts would be transformed equally. We take an alternative approach, where we promote a more complete representation of the input space. That is, different input vectors should produce different predictions, even if the corresponding predictions produce similar results. Thus, we enforce the model to reach as much leaves as possible when processing the set of input vectors of a fragment. We argue that this constraint provides a way of keeping the tree model balanced by giving in increased weight to the input space representation. We are aware that this criterion might affect the compactness of the tree model, however one has to consider this structural measure used in combination with the error-driven measure presented below.

To achieve this, we define a balance coefficient c_{bal}:

$$c_{bal} = \frac{n_{dpr}}{n_{di}} \tag{1}$$

where n_{dpr} is the number of distinct prediction leaves that were reached as a consequence of successive tests, and n_{di} is the total number of inputs vectors that differ in at least one feature.

3.2 Transformation-based error

Having in mind the structural considerations presented above, we are ultimately interested in how good the model has learned to predict expressive transformations. We define in this section the transformation-based error. As we mentionned earlier, we focus in this paper on duration, onset, and loudness deviations measured in the performed notes. From the transformation point of view, these values correspond to duration ratio, onset deviation, and loudness relative variation regarding the mean fragment loudness, averaged over all the fragment performed notes. We must design an error measurement that takes into account with equal weight each of these three dimensions. Typically, onset deviation and energy relative variation values lie around 0, while the duration ratio value lies around 1. We think an appropriate way to give the same relative weight to these three dimensions is to use the average of the mean relative error. Consequently, we define a fragment transformation error as the mean of duration ratio, onset deviation, and energy variation mean relative errors over the training fragment. We call the average of this latter error over all the training fragments Global Relative Error (GRE).

3.3 Comparing two fitness measurements

First, we compute the error-driven fitness function by applying the following formula:

$$f_{error} = \frac{1}{1 + GBE} \tag{2}$$

Based on the ideas presented in this section, we then introduce two fitness measurements:

- Fitness 1: This is the error-driven fitness function f_{error}.
- Fitness 2: This is the hybrid fitness function that integrates both error-driven and unsupervised components. We obtain it by computing the geometric mean between c_{bal} averaged over the training fragments and f_{error}.

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 4 jazz excerpts included in the *New 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 finally previous and next note relative interval. We extracted after an analysis of the acoustic recordings and an alignment with the score melodies the following expressive features: duration ratio of the performed note, onset deviation (expressed as a fraction of a quarter note) and relative note energy regarding the mean energy of the considered fragment.

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 expressive transformations that take place during the beginning of each fragment. During the test phase, we evaluate how accurately the model is able to predict the expressive transformation that take place at the end of these fragments, which we refer as unseen data. Note that we carefully checked that the test fragments do not contain melodic information present in the test set.

4.3 Results

Figure 3 shows the Global Relative Error measurements of the best-so-far tree model during the evolution. The results we present are obtained by averaging 5 different runs for each fitness setting. The legend of the figure is the following: Plain (respectively dashed) grey

plot corresponds to the results of the best-so-far model evolved with the fitness 1 and tested on the training (respectively test) cases. Plain (respectively dotted) black plot corresponds to the results of the best-so-far model evolved with fitness 2 and tested on the training (respectively test) cases.

First, we can notice an overall decrease of the GRE measured on the training data for both fitness measures. This shows that, for both fitness measures, the system is able to evolve models that minimize the error on the training data. We also notice that the models evolved with fitness 2 exhibit an error that spans a considerably larger range that the error measured on models evolved with fitness 1. Fitness 2 best-so-far model error turns to be the smallest after 70 generations, which shows that the hybrid fitness function produces a better fit on the training data. Secondly, we focus on the test error exhibited by the best-so-far models evolved with both fitness measurements. For both families of models, we identify an overfitting point in which the test error gets larger than the training error, this point is reached later in the case of models evolved with fitness 2 (generation 45) than with models evolved with fitness 1 (generation 20). Finally, even if the final test GRE is slightly larger for models evolved with fitness 2, both test errors are comparable. Obviously, these are preliminary results that should be confirmed with larger-scale experiments involving cross-validation.

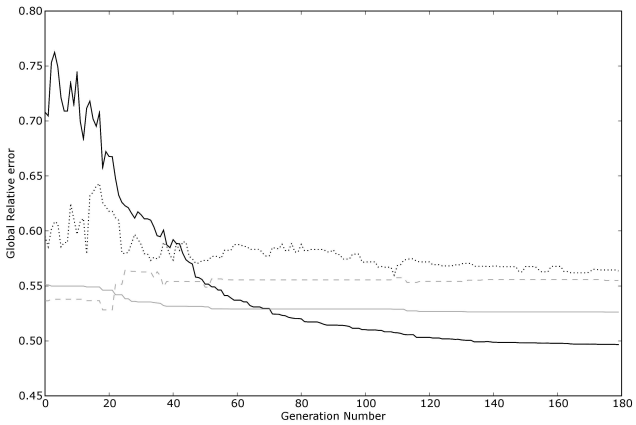

Figure 3. Global Relative Error of the best-so-far model during the evolution in four settings. Plain grey: fitness 1, training data. Dashed grey: fitness 1, test data. Plain black: fitness 2, training data. Dotted black : fitness 2, test data.

5 Conclusion

We presented in this paper an approach for building evolutionary regression trees for modelling 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 model structure and data types. Then, we defined the basis of a multi-dimensional regression tree for modelling expressive music performance. We specified the types, primitives, operators and fitness function used in the STGP framework. We finally presented preliminary results for both learning and generalization experiments, using an expressive performance database annotated from monophonic recordings of jazz standards.

We plan to extend our work in the following directions: first we plan to increase the size of the annotated performance database we are using in order to assess more robustly the training and generalization accuracy of the evolved models. Also, we plan to extend the set of possible transformations to ornamentations, consolidations and fragmentations that are characteristic of Jazz performances. 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 former expressive gestures in music performance.

6 Acknowledgments

This work is supported by the Spanish TIC project ProMusic (TIC 2003-07776-C02-01). The authors would like to thank Maarten Grachten for the useful comments he provided.

REFERENCES

[1] J. Biles, 'Genjam: A genetic algorithm for generating jazz solos', in *Proceedings of the International Computer Music Conference*, Aarhus, Denmark, (1994).

[2] Hendrik Blockeel and Luc De Raedt, 'Top-down induction of first-order logical decision trees', *Artificial Intelligence*, **101**(1-2), 285–297, (1998).

[3] M.J. Cavaretta and K. Chellapilla, 'Data mining using genetic programming: The implications of parsimony on generalization error', in *Congress on Evolutionary Computation (CEC 1999)*, pp. 1330–1337, Washington (DC), USA, (1999).

[4] P. Dahlstedt and M. Nordahl, 'Living melodies: Coevolution of sonic communication', in *First Iteration Conference on Generative Processes in the Electronic Arts*, Melbourne, Australia, (1999).

[5] C. Gagné and M. Parizeau, 'Open beagle manual. technical report rt-lvsn-2003-01-v300-r1', Technical report, Laboratoire de Vision et Systèmes numérique, Université de Laval, (2005).

[6] C. Gagné, M. Schoenauer, M. Parizeau, and M. Tomassini, 'Genetic programming, validation sets, and parsimony pressure', in *EuroGP 2006, LNCS 3905*, ed., P.Collet et al., pp. 109–120, (2006).

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

[8] 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*, Budapest, Hungary, (2006).

[9] A. Horner and Goldberg, 'Genetic algorithms and computer-assisted music composition', in *International Computer Music Conference*, Montreal, Quebec, (1991).

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

[11] David J. Montana, 'Strongly typed genetic programming', *Evolutionary Computation*, **3**(2), 199–230, (1995).

[12] Somnuk Phon-Amnuaisuk and Geraint A. Wiggins. The four-part harmonisation problem: A comparison between genetic algorithms and a rule-based system.

[13] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., 1993.

[14] Walter Alden Tackett, 'Greedy recombination and genetic search on the space of computer programs', in *Foundations of Genetic Algorithms 3*, eds., L. Darrell Whitley and Michael D. Vose, 271–297, Morgan Kaufmann, San Francisco, CA, (1995).

[15] N. Tokui and H. Iba, 'Music composition with interactive evolutionary computation', in *Proceedings of Generative Art2000, the 3rd International Conference on Generative Art*, Milan, Italy, (2000).

[16] 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).

[17] G. Widmer, 'Machine discoveries: A few simple, robust local expression principles.', *Computer Music Journal*, (2002).