Audio Engineering Society
Convention Paper

Presented at the 116th Convention
2004 May 8–11 Berlin, Germany

Automatic extraction of music descriptors from acoustic signals using EDS

Aymeric Zils[1], François Pachet[1]

[1] Sony CSL Paris, 6 rue Amyot 75005 Paris, France
{zils, pachet}@csl.sony.fr

ABSTRACT

High-Level music descriptors are key ingredients for music information retrieval systems. Although there is a long tradition in extracting information from acoustic signals, the field of music information extraction is largely heuristic in nature. We present here a heuristic-based generic approach for extracting automatically high-level music descriptors from acoustic signals. This approach is based on Genetic Programming, used to build relevant features as functions of mathematical and signal processing operators. The search of relevant features is guided by specialized heuristics that embody knowledge about the signal processing functions built by the system. Signal processing patterns are used in order to control the general processing methods. In addition, rewriting rules are introduced to simplify overly complex expressions, and a caching system further reduces the computing cost of each cycle. Finally, the features build by the system are combined into an optimized machine learning descriptor model, and an executable program is generated to compute the model on any audio signal. In this paper, we describe the overall system and compare its results against traditional approaches in musical feature extraction à la Mpeg7.

1. INTRODUCTION

The exploding field of Music Information Retrieval has recently created extra pressure to the community of audio signal processing, for extracting automatically high level music descriptors. Indeed, current systems propose users with millions of music titles (e.g. the peer-to-peer systems such as Kazaa) and query functions limited usually to string matching on title names. The natural extension of these systems is content-based access, i.e. the possibility to access music titles based on their actual content, rather than on file names. Existing systems today are mostly based on editorial information (e.g. Kazaa), or metadata which is entered manually, either by pools of experts (e.g. All Music Guide) or in a collaborative manner (e.g. MoodLogic). Because these methods are costly and do not allow scale up, the issue of extracting automatically high-level features from acoustic signals is key to the success of online music access systems.

Extracting automatically content from music titles is a long story. Many attempts have been made to identify dimensions of music that are perceptually relevant and can be extracted automatically. One of the most known is tempo or beat. Beat is a very

important dimension of music that makes sense to any listener. [1] introduced a beat tracking system that successfully computes the beat of music signals with good accuracy.
There are, however, many other dimensions of music that are perceptually relevant, and that could be extracted from the signal. For instance, the presence of voice in a music title, i.e. the distinction between instrumentals and songs is an important characteristic of a title. Another example is the perceived intensity. It makes sense to extract the subjective impression of energy that music titles convey, independently of the RMS volume level: with the same volume, a Hard-rock music title conveys more energy than, says, an acoustic guitar ballad with a soft voice. There are many such dimensions of music that are within reach of signal processing: differentiate between "live" and studio recording, recognize typical musical genres such as military music, infer the danceability of a song, etc. Yet this information is difficult to extract automatically, because music signals are usually highly complex, polyphonic in nature, and incorporate characteristics that are still poorly understood and modeled, such as transients, inharmonicity, percussive sounds, or effects such as reverberation.

2. THE TRADITIONAL METHOD

2.1. Combination of Low-Level Descriptors

Typically, the design of a descriptor extractor consists in combining Low-Level Descriptors (LLDs) as relevant characteristics of acoustic signals (features) using machine learning algorithms. More precisely, the traditional approach in descriptor design is the following (see, e.g. [2], [3], [4]):
Firstly, the signals of a reference database are labeled with the descriptor's values. These values can be obvious to get (e.g. Presence of singing voice), or can require the use of perceptive tests (e.g. the global energy of musical extracts). In this latter case humans are asked to enter a value for the descriptor, and then statistical analysis is used to find the average values considered thereafter as grounded truth.
Secondly, several features of the associated audio signals are computed. A typical reference for audio signal features is the Mpeg7 standardization process [5], that proposes a battery of LLDs for describing basic characteristics of audio signals. The purpose of Mpeg7 is not to solve the problem of extracting high level descriptors, but rather to propose a basis to design such descriptors.
Eventually, the most relevant features, i.e. that best map with the labels or values of the signals, are selected and combined into machine learning processes, to provide an optimal model for the descriptor.

2.2. Limitation of the traditional method

The traditional method sketched above works well only for relatively easy problems; problems for which generic low level features are adapted. However, generic features can only extract information which is "predominant" in the signal, and are, by definition, unable to focus on specific, problem-dependent properties. The core assumption of this paper is precisely that in order to solve more difficult problems one needs specific features adapted to the problem at hand. The following problem illustrates this claim.

A simple example: Sinus + Colored Noise

Let us consider the problem of detecting a sinus wave in a given frequency range (say 0-1000Hz) mixed with a powerful colored noise in another frequency range (1000-2000Hz). As the colored noise is the most predominant characteristic of the signal, generic features such as Mpeg7's are unable to detect the hidden sinus. For instance, when we look at the spectrum of a 650Hz sinus mixed with a 1000-2000Hz colored noise (fig.1), the peak of the sinus is visible but not predominant, and is thus impossible to extract automatically using a generic feature.

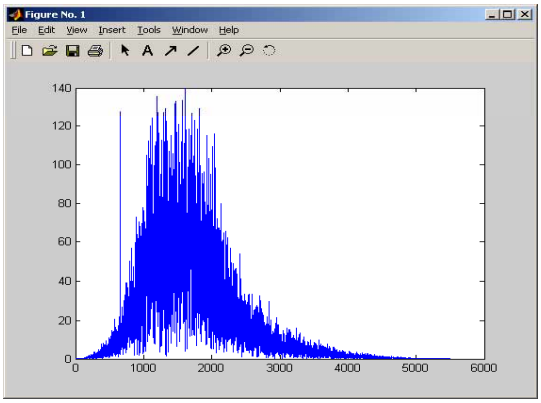

Fig 1: Spectrum of a 650Hz sinus mixed with 1000-2000Hz colored noise

Of course, this problem is easy to solve by hand, for instance by applying a pre-filtering to the signal that cuts off the frequencies of the colored noise, so that the sinus emerges from the spectrum, and becomes a predominant property (see Fig. 2).

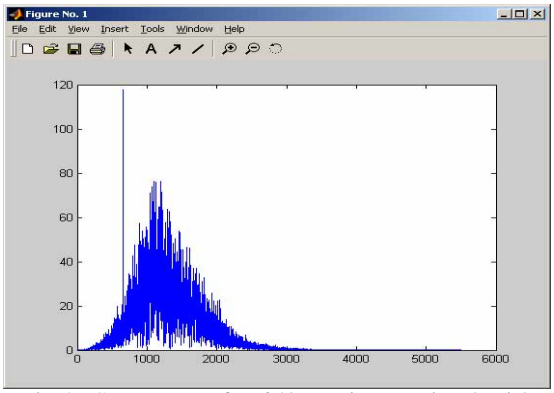

Fig 2: Spectrum of a 650Hz sinus mixed with 1000-2000Hz colored noise, pre-filtered by a 1000Hz Low-Pass Filter

This basic example illustrates the fact that combinations of basic LLDs cannot cover a function space wide enough to find specialized extractors. Indeed, we claim that high-level descriptor can be obtained by some linear combination of basic LLDs. An automatic system that produces extractors has to be able to search in a larger function space, as experts in signal processing normally do. Such a search space has to include not only actual operators but also compositions thereof as well as all the possible "in-between" processes such as filters or peak extractions inserted to improve the efficiency of an extractor.

3. IMPROVING TRADITIONAL LLD COMBINATION USING AUTOMATIC OPERATORS COMPOSITION

The design of specific features that are relevant for a given description problem is usually done by hand by signal processing experts. This section introduces the idea of generating automatically such specific features adapted to a particular problem.

3.1. Motivation for an automatic system for descriptors extraction

Although there is no known general paradigm for designing domain-specific features, their design usually follows some sort of patterns. One of them consists in filtering the signal, splitting it into frames, applying specific treatments to each segment, then aggregating all these results back to produce a single value.

This is typically the case of the beat tracking system described in ([1]), that can schematically be described as an expansion of the input signal into several frequency bands, followed by a processing of each band, and completed by an aggregation of the resulting coefficients using various aggregation operators, to yield eventually a float representing (or strongly correlated to) the tempo. The same applies to timbre descriptors proposed in the music information retrieval literature ([6], [7]) an more generally to most audio descriptors described in the literature.

Of course, this global scheme of expansion/reduction is under specified, and an infinite number of such schemes could be envisaged. Our goal is therefore to design a system that is able to 1) search automatically relevant signal processing features, seen as compositions of functions and build a model of the descriptor and 2) reduce the search space significantly using generic knowledge on signal processing operators.

3.2. Definition of a *description problem*

In the context of an automatic modeling of descriptors from numeric signals, the definition of the description problems handled by the system has to remain simple to preserve the generality of the approach. One simple way to define a description problem is to use the supervised learning approach: a set of labeled signals, also called *learning database*, defines the description problem. These labels are either numeric values, such as an evaluation of their "musical energy" (between 0 and 1), or a class label, such as the "presence of a singing voice" or not, or the genre chosen in a given taxonomy. The system will then finds the rules of the labeling of the signals, i.e. the model of the descriptor, by designing a function which produces outputs as close as possible to the learning database.

3.3. General Principle of the "Extractor Discovery System" (EDS)

The key idea of our approach is to substitute the combination of basic LLDs by the composition of signal processing operators: our system EDS composes automatically operators to discover features as signal processing functions that are optimal for a given descriptor extraction task.

The global architecture of EDS, illustrated in Figure 3, consists in two parts: modeling of the descriptor and synthesis of the extractor. Both parts are fully automatic and lead eventually to an extractor for the descriptor.

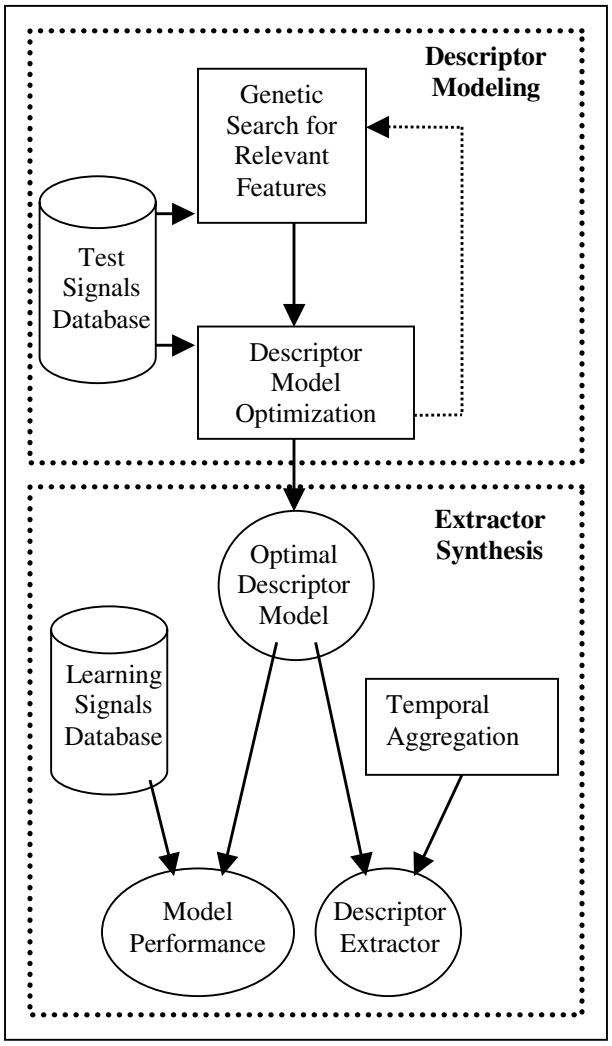

Fig. 3: Global Architecture of EDS

The modeling of the descriptor is the main part of EDS. It consists in searching automatically for a set of relevant features using the genetic search algorithm, and then to search automatically for the optimal model for the descriptor, that combines these features.

The search for specific features is based on genetic programming, a well-known technique for exploring search spaces of function compositions (see [8]). The genetic programming engine composes automatically signal processing operators to build arbitrarily complex functions.

Each built function is given a fitness value which represents how well the function performs to extract a given descriptor on a given learning database.

The evaluation of a function is very costly, as it involves complex signal processing on whole audio databases. Therefore, to limit the search, a set of *heuristics* are introduced to improve the *a priori* relevance of the created functions, as well as *rewriting rules* to simplify functions before their evaluation.

Once the system has found relevant features, it combines them to feed them into various machine learning models, and then optimizes the model parameters.

The synthesis part consists in generating an executable file to compute the best model on any audio signal. This program allows computing this model on arbitrary audio signals, to predict their value for the modeled descriptor.

4. EDS TECHNICAL DESCRIPTION

We describe here the three main ingredients of the EDS system: the automatic construction of signal processing functions, the adaptation of these functions for a given descriptor, and the combination of those into a general descriptor model.

4.1. Automatic construction of features

Functions are represented in EDS as compositions of basic operations applied on an arbitrary input audio signal. The automatic construction of correct functions relies on the control of the types of data handled by the functions, and on the introduction of signal processing expertise as heuristics.

4.1.1. Representation of functions as signal processing operators trees

The basic operators used by EDS can be mathematical, such as taking the mean values of a set, or can process a signal, temporally (such as correlation), or spectrally (such as a low-pass filtering). In addition, some operations are parameterized using constant values (like cut-off frequencies), or external signals (for example a correlation with another, fixed, reference signal).

To account for the specificity of audio extraction, we also introduced operators to implement the global extraction schemes, such as described in 3.1. For instance, the *Split* operator splits a signal

into frames, an operation that is routinely performed when a given treatment has to be made on successive portions of the signal.

The functions built by composing these operators have to contain at least one argument labeled *InSignal*, which is instantiated with a real audio signal before the evaluation of the function. Figure 4 shows an example of the syntactic representation for a function that is a composition of basic operators (FFT, Derivation, Correlation, Max):

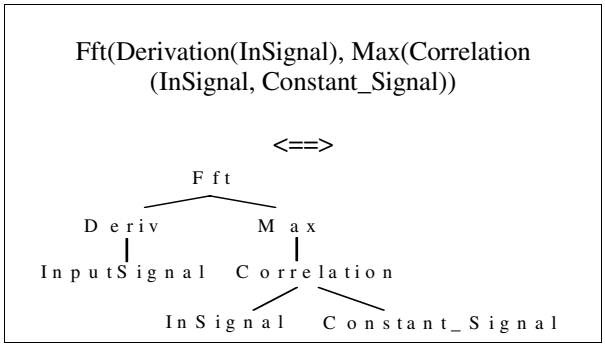

Fig. 4: The syntactic tree of a function in EDS

4.1.2. Data Types

The need for typing is well-known in Genetic Programming, to ensure that the functions generated are at least syntactically correct. Different type systems have been proposed for GP, such as strong typing ([9]) that mainly differentiate between the "programming" types of the inputs and outputs of functions.
In our context, the difference between the programming types floats, vectors, or matrix, is superficial. For example, the operator "Abs" (absolute value) can be applied on a float, a vector, etc. This homogenous view of values yields simplicity in the programming code, which we need to retain.
However, to control the physical processes in EDS, we need to distinguish how the functions built by the system handle the data, at the level of their "physical dimension". For instance, audio signals and spectrum can be seen both as vectors of floats from the usual typing perspective, but they are different in their dimensions: a signal is a time to amplitude representation, while a spectrum associates frequency to amplitude. Thus, these data have to be processed differently. Our typing system, based on the following constructs, represents this difference, to ensure that our resulting functions make sense.

Using only three physical dimensions (time "t", frequency "f", and amplitudes or non-dimensional data "a"), we are able to represent most of the data types handled by the system, by building atomic, vector, and functional types.

"Atomic" types describe the physical dimension of a single value. For instance:
- Position of a drum onset in a signal: "t",
- Cut-off frequency of a filter: "f",
- Amplitude peak in a spectrum: "a".

"Functional" types represent data of a given type, which are evolving in a dimension of another type. The evolution type is separated from the data type using the ":" notation. For instance:
- Audio signal (amplitude evolving in time): "t:a",
- Spectrum (amplitude evolving in frequency): "f:a".

"Vector" types, notated "V", are special cases of functions, used to specify the types of homogeneous sets of values without dimensional evolution. For instance:
- Temporal positions of the autocorrelation peaks of an audio signal: "Vt"
- Amplitudes of these autocorrelation peaks: "Va".

In the case of functional data with multiple evolving or vector dimensions, all the evolving types are written before the ":". For instance:
- A signal split into non-regular frames: "Vt:a",
- The autocorrelation peaks on each frame: "VVa"

4.1.3. Operators typing rules

The operations in EDS transform physically the data, and can therefore be specified using the typing system. For each operator, we define typing rules that provide the type of its output data, depending on the types of its input data. The typing rules are usually reduced into a dimensionality rule and a transformation rule.

Example 1: Absolute value
The "Absolute value" operation does not change the physical dimension of the data. Its typing rules are:
- no evolving dimension needed
- input type "T" → output type "|T|",
with |a|=a, |t|=t, |f|=f.

For instance, “Abs” transforms:
- a set of amplitudes (“Va”) into another (“Va”)
- a spectrum (“f:a”) into another spectrum (“f:a”)

Example 2: Spectrum
The “Spectrum” operation transforms a signal of type “t:a” into a frequency spectrum of type “f:a”, and transforms a frequency spectrum “f:a” into a data of type “t:a”, homogeneous with a signal. More generally, the “Spectrum” operation inverses the physical type of an evolving dimension. Its typing rules are:
- at least 1 evolving dimension
- input evolving type “T” → output evol type “T^{-1}”, with a^{-1}=a, t^{-1}=f, f^{-1}=t.

For instance, “Spectrum”:
- transforms a set of signals (“Vt:a”) into a set of spectrums (“Vf:a”)
- cannot handle temporal onsets (“Vt”)

Example 3: Split
The “Split” operation allow observing a data on regular observation windows. Thus “Split” adds a new evolving dimension, and its typing rules are:
- at least 1 evolving dimension
- input evolving type “T” → output evolving type “TT”: addition of an evolving dimension.

For instance, “Split” transforms:
- a signal (“t:a”) into a set of signals (“tt:a”)
- a set of time values (“Vt”) into multiple sets of time values (“VVt”)

This typing system is more complex than the usual typing systems used routinely in Genetic Programming, but has the interest of being able to retain the respective physical dimensions of the inputs and outputs values of all the operations in a function. For instance, the following complex but realistic function handles the following data types: Min_a ($Max_{t:a}$ ($Sqrt_{tf:a}$ ($FFT_{tf:a}$ ($Split_{tt:a}$ ($InSignal_{t:a}$))))), and thus provides as final output one amplitude value “a” from a given input signal InSignal “t:a”.

4.1.4. Controlling general processing methods using generic operators and patterns

The types of data handled by a function are a signature of the general processing methods used in the function. For instance, if an operation in the function provides data of type “f:a” (homogeneous to a spectrum), this means that the following operations are computed in the spectral domain.

In order to control globally the processing methods through the successive types of data handled by the functions, we have introduced "generic operators" that stand for one or several random real operator(s) whose output types are forced.
EDS can deal with three different generic operators (notated "*", "!", and "?") that have different functionalities:

"?_T" stands for one operation providing an output type "T".
For instance, “?_a(Signal)” can be implemented as:
- “$Max_a(Signal_{t:a})$”, or
- “$Variance_a(Signal_{t:a})$”.

"*_T" stands for a composition of several operations that all provide an output type “T”.
For instance, “*_a(Signal)” can be implemented as:
- “$Square_a\,(Max_a\,(Signal_{t:a}))$”, or
- “$Log_a\,(Square_a\,(Variance_a\,(Signal_{t:a})))$”.

"!_T" stands for a composition of several operators that provide a final output type "T".
For instance, “!_a(Signal)” can be implemented as:
- “$Variance_a\,(Autocorrelation_{t:a}\,(Signal_{t:a}))$”, or
- “$Min_a\,(Max_{t:a}\,(Fft_{tf:a}\,(Split_{tt:a}\,(Signal_{t:a}))))$”.

These generic operators allow specifying locally the processes to use in a function. By composing them to write functions *patterns, we* describe a global set of processes to apply on an audio signal to obtain a final value. For instance, the simple *pattern*
"?_a (!_Va (Split (*_t:a (Signal))))"
is a translation of the general extraction scheme presented in 3.1, standing for the following processes:
- « Apply some transformations on the input signal in the temporal domain » (*_t:a)
- « Split the resulting signal into frames » (Split)
- « Find a vector of characteristic values - 1 for each frame » (!_Va)
- « Find one operation to find one relevant characteristic value for the entire signal » (?_a)

There are various ways to instantiate this pattern, among which:
- Sum_a ($Square_{Va}$ ($Mean_{Va}$ ($Split_{Vt:a}$ ($HpFilter_{t:a}$ ($Signal_{t:a}$, 1000Hz))))), or
- $Log10_a$ ($Variance_a$ ($NPeaks_{Va}$ ($Split_{Vt:a}$ ($Autocorrelation_{t:a}$ ($Signal_{t:a}$)))))

Patterns are specified in the EDS algorithm in order to guide the search of functions. The simplest pattern to specify is "!_a", that means "a function made of any composition of operators providing one non-dimensional or amplitude value as final output".

4.1.5. Heuristics

The system is able to build physically correct functions by specifying signal processing patterns. However, the physical correctness is not sufficient to build relevant functions, by choosing the optimal operations to solve a given description problem.
In order to guide the instantiation of the patterns, we need to introduce knowledge in the system, as signal processing heuristics. Indeed, heuristics are a central point in the design of EDS. They represent the know-how of signal processing experts, about functions seen a priori, i.e. before their evaluation. The interest of heuristics is that they both favor a priori interesting functions, and rule out obviously non-interesting ones.
A heuristic in EDS associates a score to a potential composition of operators, between 0 (forbidden composition) and 10 (very recommended composition). These scores are used when EDS builds a new function, to select the candidates between all the possible operations. Basically, the heuristics allow to:

- Control the structure of the functions

For instance the number of operations done to compute the cut-off frequency argument for a high-pass filter
"HpFilter (InSignal, CutOffFreq)"
can be controlled using the heuristic
"HpFilter (Signal, Branch) => SCORE = Max (0, 5 - Size(Branch))"
The filtering operation will be scored 5 if the cut-off frequency is a constant value, 4 if it is the result of one operation, and so on.

- Avoid bad combination of operations

For instance, multiple high-pass filters are avoided using the heuristic
"HpFilter (HpFilter => SCORE = 1", labeling two consecutive high-pass filtering as a very bad composition of operations,
Similarly, filter combination rules can be translated to the following heuristics
"MpFilter(HpFilter=>3", "LpFilter(HpFilter=>5", etc.

- Range constant parameters values

For instance, the following heuristic
"Envelope (x, <50 frames) => SCORE = 1"
rules the size of the window when computing an envelope, and
"HpFilter (x, <100Hz) => 1"
rules the cut-off frequency value of a filter.

- Avoid usually useless operations

For instance the heuristic "X (X (X => SCORE = 2" avoids too many repetitions of operators:, etc.

4.1.6. Automatic construction of functions using patterns, typing rules, and heuristics

Using all the previous rules, EDS is able to build automatically various physically correct functions from a given signal processing pattern.
The pattern is composed of one input signal, and several generic operators, real operators, several numeric values, and constant signals.
The automatic synthesis of functions is performed in bottom-up fashion, starting from the input signal, and grafting sequentially the operators one after the other up to the top of the tree, all the generic operators being instantiated, i.e. replaced by real operators (as presented in 4.1.4.).

4.2. Search for optimized features

Once the system has built functions with a correct type, it evaluates them and tries to improve their fitness to solve the description problem.

4.2.1. Evaluation of function fitness

To evaluate if a function is relevant, the system computes this function on the whole learning database that defines the description problem, and then compares the values obtained with the labels of the signals, to check if the former can explain the latter. Different fitness functions can be computed, depending on the nature of the descriptor. In our experiments with regression problems, we compute fitness as the Pearson correlation coefficient between the function values and the perceptive values ([14]). For classification problems, we use the Fisher's criterion ([14]) for the function values on the different class labels, which evaluates how well the function discriminates between the different classes.

The system uses then the fitness of the functions as a criterion in the search algorithm, to find relevant functions for the problem to solve.

4.2.2. Genetic Search Algorithm

The function search part in EDS consists in building signal processing functions that are increasingly relevant, using an algorithm based on genetic programming, i.e. the application of genetic search to the world of functions, as introduced by [10].

Given a description problem for which we seek an extractor, defined by a database DB containing labeled audio signals (numeric values or class labels), the algorithm builds a population of functions from a pattern P, and tries to improve them by applying various genetic transformations on them.

More precisely, the algorithm works as follows:

1. Build the first population P0, of random functions based on the pattern P
2. Compute the fitness of each function in the population
3. If (*Stop Condition*): STOP the algorithm, and RETURN the best function
4. Else: select the functions with the highest fitness, and create of a new population P_{i+1}, by applying transformations on them
5. Iteration to (2)

The "Stop Condition" is specified using various combined criteria:

- Maximum number of iteration reached: the search stops automatically after population number 1000.
- A relevant function is found: fitness >= threshold; typically threshold=1, the function itself is a perfect model of the descriptor.
- The population does not improve anymore: fitness of the best function of population P_i = fitness of the best function of population P_{i-N}; typically N=5 unimproved populations.

Running this algorithm once provides one optimal function to be used in the final model.

Therefore, this algorithm is run N times to build N optimized functions constituting the final feature set used in the final model of the descriptor.

4.2.3. Creation of populations by genetic transformations

During the genetic search, each new population is created by applying various genetic transformations on the most relevant functions of the current population. These transformations aim at reusing local operations found in relevant functions, in order to build even more relevant functions. Three main transformations are used in EDS: structural cloning, mutation, and crossover.

Structural cloning consists in keeping the tree structure a function and applying variations on its constant parameters, such as the cut-off frequencies of filters or the computation window sizes.
For example, the function
"Sum (Square (FFT (LpFilter (Signal, *1000Hz*))))"
can be cloned as
"Sum (Square (FFT (LpFilter (Signal, *800Hz*))))".

Mutation consists in cutting a branch of a function, and replacing it by another composition of operators providing a data of the same type.
For example, the function
"Sum (Square (FFT *(LpFilter (Signal, 1000Hz)*))))",
can be mutated into
"Sum (Square (FFT *(MpFilter (Signal, 1100Hz, 2200Hz)*))))".

Finally, crossover consists in cutting a branch from a function and replacing it by a branch cut from another function.
For example
"*Sum (Square (FFT* (LpFilter (Signal, 1000Hz))))"
and "Sum (*Autocorrelation (Signal)*)"
can produce the crossover function
"*Sum (Square (FFT (Autocorrelation (Signal))))*".

In addition to the genetically transformed functions, the new population is completed with a set of new random functions to ensure its diversity and introduce new operations.

4.2.4. Improvement of the search

Eventually, in order to search for function more efficiently, rewriting rules and a caching mechanism have been included in the system.

Rewriting rules are applied to simplify functions before their evaluation, using a fixed point mechanism until to obtain a normal form. Unlike

heuristics, they are not used by the genetic algorithm to favor combinations, but they:

- Avoid computing several times the same function with different but equivalent forms.

For example:
"Correlation(x,x)==>Autocorrelation(x)", or
"HpFilter(HpFilter(x,a),b)
==>HpFilter(x,max(a,b))"

- Reduce the computation cost.

For Example: Perseval equality
"Mean(Fft(Square(x))) => Sum(Square (x))"
avoids to compute the "Fft" of a signal.

Finally, a caching mechanism is introduced to speed up the computation of functions, so that any costly function is computed once, and reused when possible.
Every time a new function is computed, all the intermediate results are stored on separate files. Finally, the most useful results are kept in memory, depending on:

- their computation time: results that require a long computation time are kept in memory,
- their utility: results that are used frequently are kept,
- their size: the allowable memory being limited, priority is given to small size results.

For instance: the computation of
"Max (Envelope (Fft (x), 100)"
will store
"x", "100", "Fft(x)", "Envelope (Fft (x), 100)", and "Max (Envelope (Fft (x), 100)" for each tested title.

4.3. Final model of the descriptor

After running the genetic search, EDS finds relevant features well adapted to the description problem at hand.
These features have now to be combined into an optimized model of the descriptor, using generic machine learning techniques (k-Nearest Neighbours, Neural Networks); the techniques can also be specific to regression (Linear Regression, Model Tree, Locally Weighted Regression) or to classification (Decision Tree, Rule Learner, Naïve Bayes, Holte's one-R, Kernel Density Classifier, Support Vector Machine, Logistic Regression, Gaussian Mixture Models). Each of these models carries with it a certain number of parameters such as the number of neighbours in the k-NN method, or the number of layers for the Neural Networks.

The processes of 1) selecting the right model and 2) finding the right parameters for this model are entirely automated in EDS.

The optimization of the model consists in a complete automatic search on all the available models for all the available parameters values. The system evaluates the performance of the models by cross-validation on the learning database, using various evaluation criteria such as the rate of good classifications, the correlation coefficient, or the kappa statistic ([14]).
The final descriptor model is the best model found, defined by:

- a set of relevant features
- a modelling technique
- optimized parameters for this technique
- a learning database DB

For instance it can be:
"DescriptorModel = KNN ("Max (Fft (InSignal))", "Variance (Autocorrelation (LpFilter (InSignal, 1000Hz)))", 6 neighbours, DB)".

The performance of this model is evaluated on a test database (different from the learning database) for assessing definitively its performance.

4.4. Self-executable extractor for the modelled descriptor

To compute the descriptor's value on a new audio signal, a executable program that computes the final model on a .wav signal is generated automatically. This program computes the values of the features, then computes the model with these values as inputs, and finally saves the result (the value of the descriptor) in an output file.

5. PERFORMANCE OF THE SYSTEM

We present here the performance on the two steps of EDS:
- Automatic synthesis of relevant features: the fitness of the best functions found indicates the capacity of the genetic search algorithm to build relevant functions regarding a given dataset.
- Descriptor modeling: the quality of the model is evaluated on a test database (see 4.3).

We compare here the results obtained by the traditional method using the Mpeg7 LLDs dataset (called "LLDs"), and by the EDS method on the following problems: basic frequency extraction, singing voice detection, musical energy modeling, and discrimination of natural sounds recorded indoor or outdoor.

5.1. Basic problem: Sinus + Colored noise

The problem consists in detecting a sinus between 10 and 1000Hz mixed with a strong colored noise between 1000-2000Hz, as described in Section 2.2.

Features

As they focus on the most predominant characteristic of the signal (the noise), the LLDs yield poor results in detecting the sinus. The best LLD, Spectral Flatness, has a correlation of 0.63 with the sinus frequencies.
After 10 populations of genetic search, EDS focuses around the function "MaxPos (FFT (LpFilter (Signal, f_c Hz)))", with different values of f_c, between 50 and 700 Hz. All these values remove efficiently the colored noise (with a Butterworth filter), and provide a correlation of 0.99. The correlation does not reach 1 because of the uncertainty near 1000Hz.

Model

The linear modeling of provides a mean prediction error of 226Hz for the LLD (Spectral Flatness), whereas it is 10Hz for the best EDS function.

These results shows that our system is able to find automatically a correct and almost optimal preprocessing with the correct parameters (here, filter ranges) to solve a simple description problem that LLDs cannot successfully address.

5.2. Subjective problem: Musical energy

The problem consists in providing a model of the subjective energy of musical extracts, based on the results of perceptive tests (see [11]). This descriptor addresses the intuitive difference there is, for example, between a punchy punk-rock song with loud saturated guitars and screaming voice conveys and an acoustic guitar ballad with a soft voice, at a constant volume level.

The tests conducted consisted in asking users to label musical extracts of various genres with the energy they "felt" while listening to the extract, independently of the listening volume. The statistical analysis of the results of these perceptive tests has shown that the musical energy is a consensual concept, and that most users feel the same energy while listening to the same songs, with a of 10 % statistical variance.

We then built a model of this "musical energy", using two labeled databases of 200 signals of length 5s at 11025Hz, one for learning, and the other for testing the performance of the model.

Features

The best LLD found was "Sum (Fft (Testwav))", with correlation=0,5418815 with the learning database labels, and 0,668248859 with the test database.
After running EDS feature genetic search algorithm 50 times, 86% of the functions created by EDS are better than the best LLD function on the learning database "Sum (Fft (Testwav))", and 46% on the test database. The best function found by EDS is "Sqrt (Percentile (Sqrt (Derivation (Sqrt (Peaks (Sqrt (Fft (Hann (LpFilter (Testwav, 5387.0)))))))), 75.0))", with correlation=0,765 on learn, and 0,812 on test.

The better performance of the features on the test database shows the generalization capacity of EDS, that has been able to find general features even though the learning database was not very well labeled.

Model

After running a forward features selection ([14]) on the LLDs, we kept 11 features to build the final LLD model of musical energy. The best method found was a linear regression with M5 selection, that provided a correlation=0.6735 on learn, and 0.7824 on test.
Running the same model with only the best EDS function provided a correlation=0.763 on learn and 0.8172 on test. Thus this only function was better than a combination of all LLDs.
After running a forward selection on the EDS functions, we kept 4 features to build the final EDS model of musical energy. The best method found was a locally weighted regression with 2 neighbors and linear weighting, that provided a correlation=0.7933 on learn, and 0.8246 on test, which corresponds to an average model error of 11.4%.

Considering the 10% variance on the perceptive tests, it can be said that EDS has provided an almost optimal model of this descriptor.

5.3. Objective classification problem: Presence of singing voice

The problem consists in providing a model that allows detecting the presence of singing voice in polyphonic audio signals, which is known as being a difficult description issue (see [12], [13]).

To compute this model, we have built two databases of 200 audio signals of length 5s at 11025Hz (for learning and testing), labeled with the 2 classes "Voice" and "No Voice".

Features

The best LLD found was also "Sum (Fft (Testwav))", with fisher=0,306 with the learning database labels, and 0,216 with the test database.
After running EDS feature genetic search algorithm 50 times, 60% of the functions created by EDS are better than the best LLD function "Sum (Fft (Testwav))", and 56% on the test database. The best function found by EDS is "Power (Abs (Sum (Integration (Integration (Fft (Sqrt (BpFilter (Sqrt (BpFilter (Sqrt (BpFilter (Testwav, 739.0, 92.0)), 238.0, 804.0)), 295.0, 1160.0))))))), 3.0)", with fisher=0,924 on learn, and 0,785 on test.

Model

After running a forward selection on the LLDs, we kept 3 features to build the final LLD model of musical energy. The best method found was a Holte's one-R with 13 objects in bucket, that provided a kappa=0.52 (76% of good classification) on learn, and 0.41 (70.5%) on test.
Running the same model with only the best EDS function provided a kappa=0.62 (81% of good classification) on learn and also 0.62 (81%) on test. Thus this only function was better than a combination of all LLDs.
After running a forward selection on the EDS functions, we kept 6 features to build the final EDS model of musical energy. The best method found was a Holte's one-R with 7 objects in bucket, that provided a kappa=0.68 (84% of good classifications) on learn, and only 0.62 (76%) on test.

The fact that the best EDS function alone provides a better test performance than the 6 best EDS functions shows that some functions generated are probably too specialized, and that a precise model built on these functions is too close to the learning database to provide good general results. Thus a balance has to be found between model precision and generalization.

5.4. Classification problem with small databases: Indoor/Outdoor sounds

The problem consists in providing a model that allows discriminating between natural sounds recorded outdoor or indoor.
To compute this model, we have built two databases of 50 audio signals of length 1s at 11025Hz (for learning and testing), labeled with the 2 classes "Indoor" and "Outdoor".

Features

The best LLD found was also "High-Frequency-Content (Testwav)", with fisher=0.224 with the learning database labels, and 0,206 with the test database.
After running EDS feature genetic search algorithm 30 times, 100% of the functions created by EDS are better than the best LLD function "High-Frequency-Content (Testwav)", on the learning and on the test database. The best function found by EDS is "Median (Holes (BpFilter (Triangle (Testwav), 312.0, 4640.0)))", with fisher=1.277 on learn, and 1,023 on test, which is far better than the best LLD.

Model

After running a forward selection on the LLDs, we kept 6 features to build the final LLD model of musical energy. The best method found was a kNN with 2 neighbours, that provided a kappa=0.85 (92.5% of good classification) on learn, but only 0.56 (78%) on test.
Running the same model with only the best EDS function provided a kappa=0.77 (88.8% of good classification) on learn, and still 0.75 (87%) on test. After running a forward selection on the EDS functions, we kept 6 features to build the final EDS model of musical energy. The best method found was a kNN with 3 neighbours, that provided a kappa=0.94 (96.3% of good classifications) on learn, and still 0.84 (92.5%) on test.

These results shows the consistency of the features found by EDS on small databases, compared to the LLDs whose models are too specific because they have to compensate the low relevancy of the features.

6. CONCLUSION

We have introduced a new approach for designing automatically efficient extractors for high-level audio descriptors. Although the proposed system, EDS, uses for the moment a limited palette of signal processing functions, it already produces results that are better than results obtained using standard manual approaches in high level descriptor extraction, in particular using the Mpg7 palette of generic features.

The generality of the approach allows EDS to address the whole class of extraction problems in the large, including the distinction between "live" and studio recording, the discrimination between simple and generic genres, the modeling of music danceability or percussivity, etc. The application of the system to non high-level extraction audio problems is also under way. Substantial increase in performance is expected by extending the palette of signal operators to more refined operators, as well as in adding more refined heuristics and rewriting rules to prune the search space.

Finally, a programming language based on EDS constructs is under way to allow users more flexibility in the definition and exploitation of the descriptors obtained.

7. REFERENCES

[1] Eric D. Scheirer. Tempo and beat analysis of acoustic musical signals. J. Acoust. Soc. Am. (JASA) 103:1 (Jan 1998), pp 588-601.

[2] Eric D. Scheirer, and Malcolm Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. Proc. ICASSP '97.

[3] P. Herrera, A. Yeterian, F. Gouyon. Automatic classification of drum sounds: a comparison of feature selection methods and classification techniques. Proceedings of 2nd International Conference on Music and Artificial Intelligence, Edinburgh, Scotland, 2002.

[4] Geoffroy Peeters, Xavier Rodet. Automatically selecting signal descriptors for sound classification. Proceedings of the 2002 ICMC, Goteborg (Sweden), September 2002.

[5] Perfecto Herrera, Xavier Serra, Geoffroy Peeters. Audio descriptors and descriptors schemes in the context of MPEG-7. Proceedings of the 1999 ICMC, Beijing, China, October 1999.

[6] JJ Aucouturier, François Pachet. Music similarity measures: what's the use ? In proceedings of the 3rd international symposium on music information retrieval (ISMIR02), Paris, October 2002.

[7] George Tzanetakis, Georg Essl, Perry Cook. Automatic musical genre classification of audio signals. Proceedings of 2nd International Symposium on Music Information Retrieval, pp 205--210, Bloomington, IN, USA, October 2001.

[8] John R. Koza. Genetic Programming: on the programming of computers by means of natural selection. Cambridge, MA: The MIT Press.

[9] David J Montana. Strongly typed genetic programming. In Evolutionary Computation 3-2, 1995, pp 199-230.

[10] David E. Goldberg. Genetic algorithms in search, optimization and machine learning. Addison-Wesley Pub. Co. 1989. ISBN: 0201157675.

[11] Aymeric Zils, François Pachet. Extracting automatically the perceived intensity of music titles. Proceedings of 6th International Conference on Digital Audio Effects (DAFX03), London, UK, September 8-11, 2003.

[12] A.L. Berenzweig, Dan P. W. Ellis. Locating singing voice segments within music signals. IEEE workshop on applications of signal processing to acoustics and audio (WASPAA01), Mohonk NY, October 2001.

[13] Wu Chou and Liang Gu, "Robust Singing Detection in Speech/Music Discriminator Design," *International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001)*, pp.865-868, Salt Lake City, Utah, USA, May 2001

[14] Fukunaga, K., "Statistical pattern recognition", Academic press, 1990.