Evolving Automatically High-Level Music Descriptors From Acoustic Signals

François Pachet, Aymeric Zils

Sony CSL Paris
6 rue Amyot, 75005 Paris, France
{pachet, zils}@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, that is used to build extraction functions as compositions of basic mathematical and signal processing operators. The search 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 function extraction methods. Rewriting rules are introduced to simplify overly complex expressions. In addition, a caching system further reduces the computing cost of each cycle. In this paper, we describe the overall system and compare its results against traditional approaches in musical feature extraction à la Mpeg7.

1 Introduction and Motivations

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. the MoodLogic). Because these methods are costly and do not allow scale up, the issue of extracting automatically high-level features from the 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. Scheirer introduced a beat tracking system that successfully computes the beat of music signals with good accuracy ([1]).

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 these information are 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.

1.1 Combining Low-Level Descriptors (LLD)

Feature extraction consists in finding characteristics of acoustic signals that map correctly with values obtained from perceptive tests. In this context, the traditional approach in designing an extractor for a given descriptor is the following (see, e.g. [2], [3], [4]):

Firstly, perceptive values are associated to a set signal of from a reference database. These values can be obvious (Presence of singing voice), or can require to conduct perceptive tests (Evaluation of the global energy of music titles): humans are asked to enter a value for a given descriptor, and then statistical analysis is applied, to find the average values, considered thereafter as a grounded truth.

Secondly, several characteristics of the associated audio signals are computed. A typical reference for audio characteristics is the Mpeg7 standardization process ([5]), that proposes a battery of LLD 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 and a format to design such descriptors.

Eventually, the most relevant LLDs are combined in order to provide an optimal extractor for the descriptor.

1.2 Two Illustrative Examples

We illustrate here descriptor extraction, using the standard approach on two music description problems, that are relevant for music information retrieval, objective, and difficult to extract automatically.

The first problem consists in assessing the perception of energy in music titles. This descriptor yields from the intuitive need for differentiating between energetic music, for instance Hard Rock music with screaming voices and saturated guitar, from quiet music, such as Folk ballads with acoustic guitar, independently of the actual volume of the music. We have conducted a series of perceptive tests on two databases of 200 titles each. For each title, we asked the listeners to rate the “energy conveyed” from "Very Low" to "Very High". We got 4500 results, corresponding to 10 - 12 answers for each title. The analysis showed that for 98% of the titles, the standard deviation of the answers was less than the distance between 2 energy categories, so the perception of subjective energy is relatively consensual, and it makes sense to extract this information from the signal. The energy is a float number normalized between 0 (no energy) and 1(maximum energy).

The second problem consists in discriminating between instrumental music and songs, i.e. to detect singing voice. The technical problem of discriminating singing voice with speech in from a complex signal is known to be difficult ([6], [7]) and remains largely open. No experiment were performed for this descriptor as the values are obvious to assess. The presence of voice is a Boolean value 0 (instrumental) or 1(song).

1.3 Results Using Basic LLD

We present here the results obtained on our two problems, using the standard approach sketched above. More precisely, the palette of LLD used for our experiments consisted of 30 LLD, obtained as Mean and Variance of:

Amplitude Signal, Amplitude Fft, High freq content, Max spectral freq, Ratio high freq, RMS, Spectral Centroid, Spectral Decrease, Spectral Flatness, Spectral Kurtosis, Spectral Roll Off, Spectral Skewness, Spectral Spread, Total Energy, Zero Crossing Rate. The method consisted in finding the linear combination of LLD that best matches the perceptive results.

First, the optimal combination is computed on each learning database: for the Global Energy problem, the regression consists in minimizing the average model error compared to the perceptive results; for the Instrumental/Song problem, the classification consists in maximizing the discrimination and finding a threshold to separate the 2 classes.

Then the combination is tested on each test database: for the Global Energy problem, the evaluation consists in computing the average model error compared to the perceptive results; for the Instrumental/Song problem, the evaluation consists in computing the recognition rate.

A cross-validation ensures the consistency of the method. The final results presented here are the mean results of the cross-validations with their uncertainty:

Table 1. Results obtained using basic Mpeg7 LLD combination on two high-level description problems

	SUBJECTIVE ENERGY (Model Error)	PRESENCE OF VOICE (Recognition Rate)
BEST FEATURE	16.87% +- 1.48%	61.0% +-10.21%
BEST COMBINATION	12.13% +-1.97%	63.0% +-11.35%

The success of the standard approach is dependent on the nature and quality of the basic signal extractors in the original palette. Mpeg7 provides some interesting descriptors, in particular in the field of spectral audio, but to extract complex, high-level musical features, the features have to be improved with additional operations, as shown in the basic example below.

1.4 Limitations of the traditional method

Example: Sinus + Colored Noise.
Let us consider the simple problem of detecting a sinus wav 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, traditional features focus on it and are unable to detect the 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 very hard to extract automatically.

However this problem is very easy to solve by hand, by applying a pre-filtering that cuts off the frequencies of the colored noise, so that the sinus emerges from the spectrum. As seen on Fig.2, the sinus peak emerges when the signal is low-pass filtered, and is thus very easy to extract automatically.

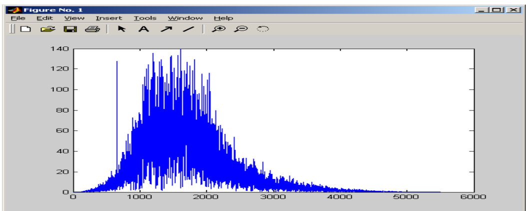

Fig. 1. Spectrum of a 650Hz sinus mixed with 1000-2000Hz colored noise

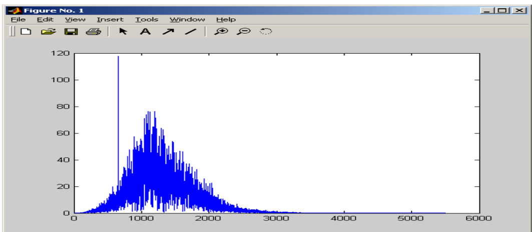

Fig. 2. Spectrum of a 650Hz sinus mixed with 1000-2000Hz colored noise, pre-filtered by a 1000Hz Low-Pass Filter

Motivations for EDS
This basic example shows that the combination of basic LLD does not cover a function space wide enough to find specialized extractors. It is not the case that any high-level descriptor can be obtained by some linear combination of basic LLD. So an

automatic system that produces extractors should be able to search in a larger and more complex function space, as experts in signal processing normally do. The variations concern not only the actual operators used in the whole process, but also their parameters, and the possible “in-between” process such as filters, peak extractors, etc... that can be inserted to improve the efficiency of the extractor.

Although there is no known general paradigm for extracting relevant descriptors, the design of high-level extractors usually follow regular patterns. One of them consists in filtering the signal, splitting it into frames, applying specific treatments to each segments, 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 treatment 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 timbral descriptors proposed in the music information retrieval literature ([8], [9]). Of course, this global scheme of expansion/reduction is under specified, and a virtually infinite number of such schemes can be searched. Our motivation is to design a system that is able to use a given signal-processing knowledge, such as patterns or heuristics, in order to searches automatically signal processing functions specialized in feature extraction. The next Section presents the design of the system.

2 EDS: From Low-Level Descriptors Combination to Signal Processing Operators Composition

The key idea of our approach is to substitute the combination of basic LLD by the composition of operators. Our Extraction Discovery System (called EDS) aims at composing automatically operators to discover signal processing functions that are optimal for a given descriptor extraction task.

The core search engine of EDS is based on genetic programming, a well-known technique for exploring functions spaces [10]. The genetic programming engine automatically composes operators to build functions. Each function is given a fitness value which represents how well the function performs to extract a given descriptor; this is typically the correlation between the function values and the perceptive values. The evaluation of a function is therefore very costly, as it involves complex signal processing on whole audio databases. To guide the search, a set of *heuristics* are introduced, to control the creation of functions, as well as *rewriting rules* that simplify functions before their evaluation. This section presents EDS design principles.

2.1 Representation of Basic Signal Processing Operators

Each operator is defined by its name, its output type, and an executable program, which evaluate the function once it is instantiated. In EDS, these programs are written and compiled in Matlab. EDS functions include constants, mathematical operators such as mean or variance, signal processing operators, temporal such as correlation, or spectral such as FFT or filters. To account for the specificity of audio extraction, we introduced operators to implement the global extraction schemes. 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.

Functions are built by composing these operators, each function containing at least one argument labeled *InSignal*, which is instantiated with a real audio signal before evaluation. Fig.3 shows an example of tree representation for a function that is a composition of basic operators (FFT, Derivation, Correlation, Max):

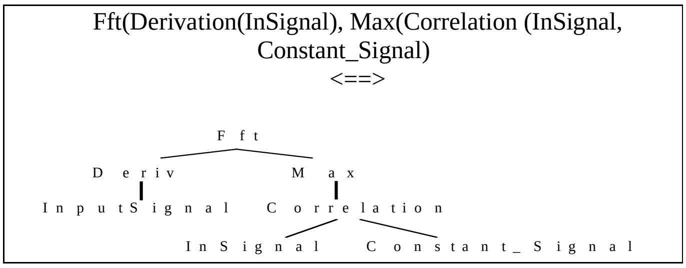

Fig. 3. Tree representation of a signal processing function

2.2 Data and Operators 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 ([11]), that mainly differentiate between the "programming" types of the inputs and outputs of functions.

In our context, the difference between 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, that we need to retain. However, we need to distinguish functions, at the level of their "physical dimension". 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. Our typing system, based on the following constructs, has to represent this difference, to ensure that our resulting functions make sense.

Atomic types, functions, vectors

Types can be either atomic dimensions, which are of 3 sorts: time, notated "t", frequency "f", and amplitudes "a". These 3 types allow to build more complex types: functions and vectors.

Functions are representations from one dimension to another. Their type is represented using the ":" notation, which differentiates between the x and y-axis of the representation. For example, the type of an audio signal (time to amplitude representation) is "t:a", whereas the type of a spectrum (frequency to amplitude) is "f:a".

Vectors are special cases of functions, associating an index to a value. Because vectors are very frequent, we introduce the shortcut symbol "V + data type" to denote a vector. For instance, a list of time onsets in an audio signal is notated "Vt", or the type of a signal split into frames is "Vt:a".

Typing rules

The output types of the operators are computed dynamically in a bottom-up fashion and recursively according to specific typing rules depending on the type of the input data. For instance, in the case of vectors, the transfer rule is: "Type (F(Vx)) = V Type(F(x))".

For non-vector arguments, each operator defines a specific typing rule. For instance:

- the output type of Abs is the type of its input: Type (Abs(arg)) = Type(arg)
- the FFT operator multiplies the x-axis dimension of its input by -1: Type (FFT (a:b)) = a^{-1}:b, thus transforms "t:a" into "f:a", and reversely "f:a" into "t:a"
- spliting the data introduces a vector of the same type: Type (Split (x)) = V Type(x),
- and so forth...

This typing system is more complex than the usual typing systems used routinely in GP, but has the interest of being able to retain the respective physical dimensions of the inputs and outputs values of functions. For instance, given an input signal S, the following complex (but realistic) function gets the following type:

Type (Min(Max(Sqrt(Split(FFT(Split (SIGNAL, 3, 100)), 2, 100)))) = "a"

Generic Operators & Patterns

This typing system allows to build "generic operators" that stand for one or several random operator(s) whose output type (and also possible arguments) are forced. 3 different generic operator (notated "*", "!", and "?") have different functionalities:

- "?_T" stands for 1 operator whose output type is "T"
- "*_T" stands for several operators whose output type are all "T"
- "!_T" stands for several operators whose only final output type is "T"

These generic operators allow to write functions patterns, that stand for any function satisfying a given signal processing method. For instance, the pattern "?_a (!_Va (Split (*_t:a (SIGNAL))))" stands for:

- « Apply some signal transformations 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)

This is the general extraction scheme presented in 1.4, it can be instantiated as:

- Sum_a (Square_Va (Mean_Va (Split_Vt:a (HpFilter_t:a (SIGNAL_t:a, 1000Hz), 100)))), or
- Log10_a (Variance_a (NPeaks_Va (Split_Vt:a (Autocorrelation_t:a (SIGNAL_t:a), 100), 10)))

Patterns can be specified in the EDS algorithm to guide the search of functions.

2.3 EDS Algorithm

The global architecture of the system consists in 2 steps (see Fig.4):

- Learning of relevant features using a genetic search algorithm,
- Validation of features and synthesis of a descriptor extractor using features combination.

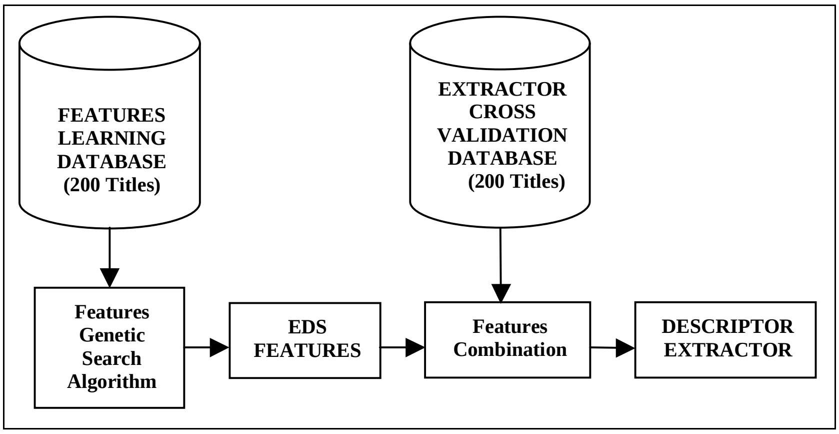

Fig.4. EDS Global Architecture

EDS features search algorithm is based on genetic programming, i.e. the application of genetic search to the world of functions, as introduced by Goldberg ([12]). More precisely, the algorithm works as follows, given:

- A descriptor D for which we seek an extractor, and its type (currently either “Boolean” or “Float”)
- A database DB containing audio signals
- A result database containing the result of the perceptive test for the descriptor D for each signal in DB

Global algorithm

The algorithm proceeds as follows:

- Build the first Population P0, by computing N random signal processing functions (compositions of operators), whose output type is compatible with the type of D.
- Begin Loop:
    - Computation of the functions for each audio signal in DB,
    - Computation of the fitness of each function, for instance the correlation between its values on DB and the associated perceptive values

- if the (fitness >= threshold) or (max number of iterations reached), STOP and RETURN the best functions
- Selection of the most correlated functions, crossover and mutation, to produce a new population P_{i+1}
- Simplification of the population P_{i+1} with rewriting rules
- Return to Begin Loop

Creation of populations

To create initial populations, a random function generator creates functions according to a given pattern (see 2.2). The generator works bottom-up, starting with the audio Signal, and finding successively operators that accept the current operator as input. New populations are then computed by applying genetic operations to the most relevant functions of the current population, that are structural cloning (constants variations), mutation, and crossover.

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

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

Crossover consists in cutting a branch in 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))))".

Eventually, to ensure diversity , new populations are completed with a set of new random functions.

2.4 Heuristics

Heuristics are vital ingredients to guide the search and 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 between 0 and 10 to a potential composition of operators. These scores are used by EDS to select candidates at all the function creation stages (random, mutation, cloning and crossover). Here are some examples of important heuristics:

- To control the structure of the functions: "HpFilter (Signal, Branch) => SCORE = Max (0, 5 - Size(Branch))". This heuristic limits the complexity of computation of arguments such as filters cut-off frequencies.
- To avoid bad combination of filters: "HpFilter (HpFitler => SCORE = 1", "Mp (Hp=> 3", "Lp (Hp => 5".

- To range constant values: "Enveloppe (x, <50 frames) => SCORE = 1" ; "HpFilter (x, <100Hz) => 1", etc...
- To avoid usually useless operations: "X (X (X => SCORE = 2" (too many repetitions of operators), etc...

2.5 Rewriting rules

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:

- 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.

2.6 Caching

Finally, to speed up the computation of functions, a caching mechanism is introduced, 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. For instance: "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.

The caching technique consists in keeping in memory the most useful results, 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.

3 Results

We present here the results of the 2 steps of EDS:

- Features computation (learning results): the correlation of the best functions found by the system evaluates how our genetic search algorithm is able to build relevant functions regarding a given data set. Correlations are computed on the whole features learning database.
- Descriptor extraction (test results): final model error (regression) or final recognition rate (classification) for 1 or a combination of the N most relevant functions. Performances and errors are evaluated using cross-validation on an independent test database, so the results are given with an uncertainty that corresponds to the cross-validation variations. For each of these descriptor, we

compare the results obtained by the traditional LLD method, the EDS method, and a combination of both.

3.1 Sinus + Colored noise (Basic problem)

The problem consists in detecting a sinus between 10 and 1000Hz mixed with a strong colored noise between 1000-2000Hz (see 1.4).

Best Features and Extractors

As they focus on the most predominant characteristic of the signal (the noise), the LLDs have poor results in detecting the sinus. The best LLD, Spectral Flatness, has a correlation of 0.63 with the sinus frequencies.

EDS focuses after 10 populations around the function "MaxPos (FFT (LpFilter (Signal, f_c Hz)))", with different values of f_c. Values between 50 and 700 Hz, that most efficiently remove the colored noise (with a Butterworth filter), provide a correlation of 0.99. The correlation does not reach 1 because of the uncertainty near 1000Hz. For the Spectral Flatness, the mean prediction error is 226Hz, whereas it is 10Hz for the best EDS function.

3.2 Perceived Intensity (Regression problem)

The problem consists in providing a model of the subjective energy of musical extracts, based on the results of perceptive tests (see 1.3). For comparison, note that a random feature has typically a correlation of 0.03, and its best combination provides a model error of 21% (the extraction function is a constant value, that is the mean value of the energies of all the titles in the database).

Best Features and Extractors

The best LLD has a correlation of 0.53 with the perceptive values, and provides a model error 16.9%+-1.5%. The best EDS feature has a correlation of 0.68 and provides a model error 14.5%+-1.8%. The best combination of LLD provides a mean model error of 12.1%+-1.9%. Adding the best EDS features decreases the mean model error to 11.4%+-1.9%.

Table 2. Model Errors for the Subjective Energy Problem

METHOD	RANDOM	BEST LLD	BEST LLDs COMBINATION	BEST EDS	COMBINATION LLDs + EDSs
PERCEIVED ENERGY (MODEL ERROR)	21%	16.9%	12.1%	14.5%	11.3%

3.3 Presence of Voice (Classification problem)

The problem consists in providing an extractor that detects the presence of singing voice (see 1.3). Note that a random feature has typically a correlation of 0.05, and its best combination provides a recognition rate of 50% (the extraction function is a constant value, that assigns the same result to all the titles in the database).

Best Features and Extractors

The best LLD has a correlation of 0.28 with the perceptive values, and has a recognition rate of 61.0%+-10.2%. The best EDS feature has a correlation of 0.54, and provides a recognition rate of 73.5%+-9.4%. The best combination of LLD provides a recognition rate of 63.0%+-11.4%. Adding the best EDS features increases the recognition rate to 84.0%+-7.7%.

Table 3. Recognition Rates for the Presence of Singin Voice Problem

METHOD	RAND	BEST LLD	BEST LLDs COMBINATION	BEST EDS	COMBINATION LLDs + EDSs
PRESENCE OF SINGING VOICE (RECOGNITION RATE)	50%	61%	63%	73.5%	84.5%

4 Conclusion

We have introduced a new approach for designing high level audio feature extractors, based on genetic programming. The proposed system, EDS, uses for the moment a limited palette of signal processing functions. However, EDS produces results that are comparable (as good or best) to standard manual approaches in high level descriptor extraction. Substantial increase in performance should be obtained by extending the palette of signal operators to more refined operators. New heuristics will also be found by analyzing the application of EDS to other high level descriptor problems, such as the distinction between “live” and studio recording), the discrimination between simple and generic genres (such as military music, music for children, etc.), or the danceability. Finally better fitness method can be used, including in particular a fully-fledged learning mechanism to match optimally the outputs of the functions to perceptive tests.

References

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

2. [Scheirer. and Slaney 1997] Eric D. Scheirer, and Malcolm Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. Proc ICASSP '97, pp. 1331-1334.
3. [Herrera & al, 2002] 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. [Peeters & al, 2002] Geoffroy Peeters, Xavier Rodet. Automatically selecting signal descriptors for sound classification. Proceedings of the 2002 ICMC, Goteborg (Sweden), September 2002.
5. [Herrera & al, 1999] 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. [Berenzweig & al, 2001] 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.
7. [Chou & Gu, 2001] 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
8. [Aucouturier & al, 2002] 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.
9. [Tzanetakis & al, 2001] 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.
10. [Koza, 1992] John R. Koza. Genetic Programming: on the programming of computers by means of natural selection. Cambridge, MA: The MIT Press.
11. [Montana, 1995] David J Montana. Strongly typed genetic programming. In Evolutionary Computation 3-2, 1995, pp 199-230.
12. [Goldberg, 1989] David E. Goldberg. Genetic algorithms in search, optimization and machine learning. Addison-Wesley Pub. Co. 1989. ISBN: 0201157675.