Evolving a Multi-Classifier System with Cartesian Genetic Programming for Multi-Pitch Estimation of Polyphonic Piano Music

Rolando Miragaia
CIIC, School of Technology and Management, Polytechnic of Leiria
Portugal
rolando.miragaia@ipleira.pt

Francisco Fernandez de Vega
University of Extremadura
Mérida, Spain
fcofdez@unex.es

Gustavo Reis
CIIC, School of Technology and Management, Polytechnic of Leiria
Portugal
gustavo.reis@ipleiria.pt

ABSTRACT

This paper presents a new method for multi-pitch estimation on piano recordings. We propose a framework based on a set of classifiers to analyze the audio input and identify the piano notes present on the given audio signal. Our system's classifiers were evolved using Cartesian Genetic Programming: we take advantage of Cartesian Genetic Programming to evolve a set of mathematical functions that act as independent classifiers for piano notes. Our latest improvements are also presented, including test results using F-measure metrics. Our system architecture is also described to show the feasibility of its parallelization and implementation as a real time system. The proposed approach achieved competitive results, when compared to the state of the art.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

Genetic Programming, Multi pitch Estimation, Automatic piano music transcription, piano notes detection

ACM Reference Format:
Rolando Miragaia, Francisco Fernandez de Vega, and Gustavo Reis. 2021. Evolving a Multi-Classifier System with Cartesian Genetic Programming for Multi-Pitch Estimation of Polyphonic Piano Music. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3412841.3441927

1 INTRODUCTION

Multi-pitch estimation is the process of identifying the musical notes (pitches) on polyphonic audio. It consists on estimating the pitch values of all concurrent sound sources (musical instruments) at each individual time frame and plays a fundamental role on the process of Automatic Music Transcription (extracting the instrument's score of a given music or audio signal). Automatic Music Transcription is a very difficult problem from both musical point of view and computational point of view: although there has been a lot of research devoted to it, it still remains unsolved. To tackle this problem, we have considered evolutionary algorithms, in particular, Genetic Programming (GP) methodology. Among all the GP variants available, we decided to innovate and use Cartesian Genetic Programming (CGP), which was first presented by Miller in [14] as a general form of genetic programming. This form of Genetic Programming is called "Cartesian" because it represents a program using a two dimensional grid of nodes.

Cartesian Genetic Programming has already proved its abilities for synthesizing complex functions capable of extracting main features from images and performing image segmentation [5]. Given its success on image processing, we decided to apply CGP on audio processing, with the objective of detecting and recognizing piano notes or pitches on polyphonic sounds. We started our research with this goal in mind and, to address this problem, and also future problems related to audio signal processing, we created a CGP toolbox for Matlab [15]. With the help of this toolbox, we defined an architecture for a classification system based on CGP. Our proposed system is composed by multiple and independent classifiers each one containing an evolved mathematical expression with multiple mathematical functions and filters and is first described in [6]. The evolved classifiers resulting from the CGP system are capable of identifying the presence of any piano key in a polyphonic sound sample and work independently: they can run in parallel. This is the primordial task of an automatic music transcription system, that can be used on a sequential frame based approach or using onset and offset detection for note tracking.

This paper is the result of our continuous research on multi-pitch estimation of piano music and describes our current architecture and special features as well as our results, in what concerns to accuracy and time performance. Thus, this paper explains the latest achievements in our CGP system due to the continuous improvements as well as the new features and techniques added that lead us to significantly improved results. The rest of this paper is organized as follows: Section 2 explains the related work. Section 3 presents the Cartesian Genetic Programming features, Section 4 explains our approach and on Section 5 we show our experiments and results. Finally, on Section 6 we present our conclusions and future work.

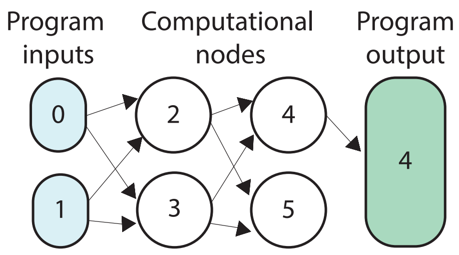

Figure 1: Overall structure of a CGP program. Program inputs and computational nodes are numbered sequentially. The program outputs can link to any computational node or program input.

2 RELATED WORK

Multi-pitch estimation has been addressed by several researchers using different approaches. Klapuri [7], proposed an iterative approach algorithm, based on harmonicity and spectral smoothness. Marolt [10], proposed a connectionist approach using a network of adaptive oscillators to detect periodicities on the audio signal, along with a partial tracking technique based on an auditory model, which converts the acoustic signal into time-frequency space. Yeh et al. [22] presented an algorithm based on the short-time Fourier transform (STFT) representation, using an adaptive noise level estimation algorithm with harmonic matching. Reis et al. [19], used a genetic algorithm approach which relies on an adaptive spectral envelope modeling and dynamic noise level estimation. Benetos and Weyde [16], based on probabilistic latent component analysis and supporting the use of sound state spectral templates, proposed an efficient, general-purpose model for multiple instrument polyphonic music transcription. However, currently state of the art approaches still fall behind the most skilled musicians. Multi-pitch estimation, is a problem that yet remains to be solved. Also, among the evolutionary approaches in the literature [19–21], real-time processing is beyond the scope of those algorithms. Which lead us to the following motivation: improve results, but also to devise a solution that allows evolutionary algorithms to reach real-time. Thus, being competitive and useful for commercial applications.

3 CARTESIAN GENETIC PROGRAMMING

Cartesian Genetic Programming was proposed by Julian Miller in 2000 [14] and is an efficient form of Genetic Programming [8, 9] with increasing popularity. In its classic form, it uses a very simple integer notation to represent a program in the form of a directed graph. Graphs are very useful program representations and can be applied to many domains like, for example, electronic circuits and neural networks. CGP Programs have three major components: program inputs, computational nodes and program outputs, as depicted in Figure 1. The genotype is a list comprised mostly of integers that represent the program primitives and how they are connected together (see Figure 2). Programs are represented as graphs in which there are non-coding genes. We use three different types of genes:

- connection genes, which are employed for connections between nodes inputs and outputs;
- function genes, which are in charge of specifying the functions from a function set;
- additional parameters: required by the functions.

This representation is very simple, flexible and useful for many types of problems. The general form of a CGP graph is shown in Figure 2. Typically, all functions have as many inputs as the maximum function arity and unused connections are ignored.

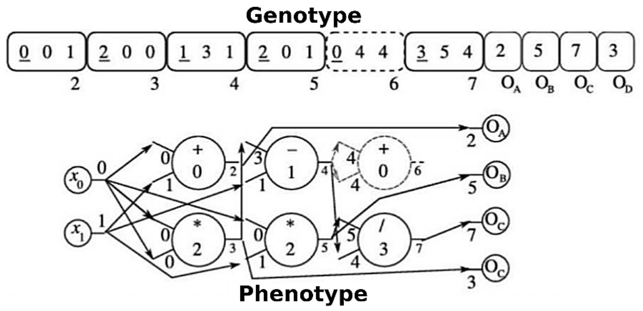

Figure 2: A CGP exmaple: Genotype and corresponding schematic Phenoty. It is a grid of nodes connected as a graph whose functions are chosen from a set of primitive functions. There are 2 inputs and 4 outputs. The grid has $n_c = 3$ (columns) and $n_r = 2$ (rows).

CGP is called "Cartesian" because it considers a grid of nodes that are addressed in a cartesian coordinate system. Each node may contain additional genes for encoding additional parameters that might be necessary for specific functions, like, for instance, a threshold value. As in any evolutionary algorithm, each individual encodes a possible solution to the problem being addressed. The set of individuals is called the population. To evaluate the quality of each individual or solution, a fitness function is used. This way, all the solutions among the population are evaluated and the current best solution is found. The next population (next generation) is then generated, based on the current best individual. The next generation contains a new set of possible solutions and becomes the current population. This process is repeated over and over, from iteration (generation) to iteration. This way, the quality of the population improves (evolves) from generation to generation, pursuing the best solution to the problem.

Algorithm 1 General CGP Algorithm

```
1: Generate initial population at random (subject to constraints)
2: while stopping criterion not reached do
3:     Evaluate fitness of genotypes in population
4:     Promote fittest genotype to new population
5:     Fill remaining places in the population with mutated versions of
   the fittest
6:     Return to step 2 until stopping criterion reached
7: end while
```

CGP algorithm, shown in Algorithm 1, begins with the generation of the initial population. Then, a fitness function is used to evaluate the quality of each individual in the population. The evolutionary strategy chooses the fittest one (best individual) and

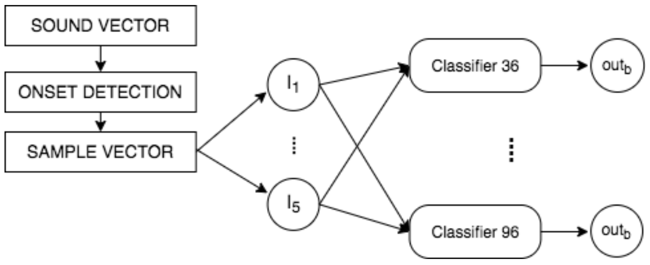

Figure 3: System workflow: 5 inputs extracted from audio sample signal, and 61 classifiers working in parallel generating 61 binary outputs

promotes it directly to the next generation. The remaining places in the population are filled with mutated versions of the fittest individual. The algorithm stops when the stopping criterion is reached.

4 PROPOSED CGP SYSTEM

A CGP encoded individual, in its general form, consists of a grid of nodes, where each node has connections and also a function chosen from a set of primitive functions. The grid has n_c (columns), n_r (rows) and *levels-back*: how many previous columns of cells may have their outputs connected to a node in the current column (see Figure 2). Depending on n_r, n_c and *levels-back*, a wide range of graphs can be generated. When $n_r = 1$ and *levels-back* $= n_c$, arbitrary directed graphs can be created with a maximum depth. Choosing these parameters imposes the least constraints. Therefore, this is the best and most general choice [13].

In our system, each piano note is identified by a CGP evolved classifier (see Figure 3). This way, for identifying 61 musical notes (from the C2 to the C7), we need to have 61 evolved classifiers: one for each musical note or piano key. Each one of these classifiers uses several inputs, all of them deriving from an acquired audio signal and returns one binary output, indicating if the corresponding piano note is present or not in the given signal. Basically, a sound vector is sampled, using the onset detector proposed by Martins [11] with some improvements [19]. Then, 5 inputs are computed from the original sound vector using several signal processing techniques. These inputs are then used by the classifiers to accomplish an output vector. This vector suffers a binarization process to obtain a final binary output.

4.1 Training

To develop a classifier system, the first stage consists in the training of the system classifiers. The training process of our system is described in Figure 4: first, some preprocessing (see section 4.2) is applied on the input audio signal to generate the inputs for all the classifiers; these inputs, a function set and a set of parameters are used in the CGP block, where the classifiers are evolved independently using the CGP toolbox; during this evolutionary process, each classifier generates an output vector; this output vector suffers a binarization process to obtain a final binary output; then, a fitness function is computed to evaluate the quality of the corresponding classier using F-measure. The engine of the implemented evolutionary process is the CGP block: it is responsible for the evolutionary

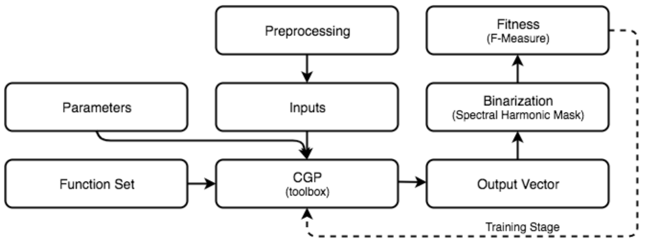

Figure 4: System architecture

process that is done during the training stage to accomplish one final classifier: a graph of mathematical functions, capable of detecting the presence of the sound produced by a piano key. This task was implemented using our Matlab Toolbox: CGP4Matlab [15]. This toolbox was developed for solving signal and image processing problems with CGP, it is open source and available for general use.[1]

Many decisions and processes had to be made and implemented besides the CGP. We had to apply some preprocessing (section 4.2) to the system inputs, had to figure out which inputs to use and how to represent them (section 4.3). We also had to create a gene encoding (section 4.4). Then, we had to decide what kind of mutations (section 4.5) and which evolutionary strategy (section 4.6) to use as well. We also developed a binarization strategy (section 4.7) for the output and fitness (section 4.8) computation.

4.2 Preprocessing

One important decision in designing classifier systems is defining the inputs and their structure.

The original sound signals acquired from wav files are float vectors, representing the input audio signals in time domain. However, the frequency domain contains a lot of important information for multi-pitch estimation problems.

As mentioned by Yeh et al. [22], the polyphonic signals can also be expressed as a sum of harmonic sources plus a residual[2]:

$$x[n] = \sum_{m=1}^{M} x_m[n] + z[n], M > 0 \text{ with } x_m[n] \approx x_m[n + N_m] \quad (1)$$

where n is the discrete time index, M is the number of harmonic sources, $x_m[n]$ is the quasi-periodic part of the mth source, N_m represents the period of the mth source and $z[n]$ is the residual. Thus, in frequency domain we have the information we need about the Fundamental Frequencies (F0) that compose the signal. This stresses for a need of converting the input audio signal to the frequency domain. This transformation is done by employing the Discrete Fourier Transform (see Equation 4).

The piano input signals are vectors in discrete time, sampled at a rate of 44100 samples per second. For the preprocessing task, we split each piano sound signal into frames of 4096 samples width, corresponding to 93 milliseconds of sound. Those frames can be acquired manually or according to a criteria, such as a threshold or onset detector.

[1]Github download link: https://github.com/tiagoinacio/cgp4matlab
[2]The residual - $z[n]$ - comes from components that are not explained by the sinusoids, for instance, the background noise, spurious components or non-harmonic partials.

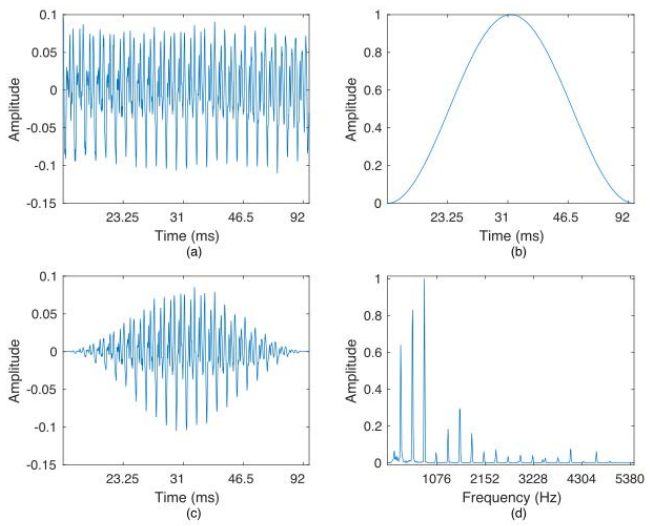

Figure 5: preprocessing process: (a) input time signal, (b) Hanning window, (c) resulting windowed signal (d) frequency domain signal after DFT

Each audio frame is then windowed (see Equation 2) using an Hanning window function (see Equation 3) to avoid spectral leakage. The windowing process is illustrated in Figure 5.

$$x_w[n] = w[n].x[n]. \tag{2}$$

$$w[n] = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right). \tag{3}$$

The windowed audio frames are transformed from the time domain to the frequency domain using the Discrete Fourier Transform (DFT):

$$X[k] = \sum_{n=0}^{N-1} x_w[n] e^{-j(\frac{2\pi}{N})nk}, (k = 0, 1, \cdots, N-1), \tag{4}$$

In discrete time processing, the DFT is typically computed using the Fast Fourier Transform algorithm (FFT), which is much faster. We obtain the signal in the frequency domain $X[k]$, using Equation 4 where where $x_w[n]$ is the time signal windowed and N is the number of samples. Any time signal cannot be uniquely represented for frequencies above $\frac{f_s}{2}$ (also known as the Nyquist frequency), where f_s is the sampling frequency of the sequence. Due to the periodicity of frequency domain signal resulting of a DFT, period $[0; f_s]$, and the Nyquist theorem where the symmetry of the real part and the antisymmetry of the imaginary part relative to Nyquist frequency, $\frac{f_s}{2}$, we may use half of the resulting signal of the DFT. Thus, from the resulting frequency signal representing in the frequency interval $[0; f_s]$ with size of 4096, we can use only the first half represented in the interval $[0; \frac{f_s}{2}]$, with a 2048 length. The preprocessing process is illustrated in Figure 5.

4.3 Inputs

As depicted in Figure 3, our system uses 5 input vectors, all acquired from the sample vector. The vector $X[k]$ resulting from Equation 4, is in the frequency domain and is a vector of complex numbers. This lets us use two different vector representations of complex numbers (z): cartesian (Equation 5) and polar (Equation 6).

$$z = a + bi \tag{5}$$

$$z = r(\cos\theta + i \sin\theta) \tag{6}$$

This way, we have a pair of vectors for each representation (2 components), making 4 usable inputs a, b, r and θ. An additional 5th input was added to the system: the signal cepstrum [18]. The cepstrum is a mathematical transformation used in audio signal processing, in particular for period estimation. Cepstral analysis is one of several methods that enables us to find out whether a signal contains periodic elements. This way, cepstral information can also be used to determine the pitch of a signal [17]. Cepstrum is defined as the inverse DFT of the log magnitude of the DFT of a signal:

$$c[n] = F^{-1}\{log|F\{x[n]\}|\}. \tag{7}$$

For a windowed frame audio signal $x[n]$ cepstrum is:

$$c[n] = \sum_{n=0}^{N-1} log(|\sum_{n=0}^{N-1} x[n]e^{-j(\frac{2\pi}{N})nk}|)^{j(\frac{2\pi}{N})nk}, (k = 0, 1, \cdots, N-1). \tag{8}$$

The cepstral coefficients describe the periodicity of the spectrum. A peak in the cepstrum denotes that the signal is a linear combination of multiples of the frequency pitch. The pitch period can be found as the number of the coefficient where the peak occurs.

Regarding the importance of the cepstral information in pitch estimation, we use it as our 5th system input. Figure 6, shows the 5 system inputs: four of them are directly obtained from the DFT of the original input audio signal: real (a), imaginary (b), radius (r) and angle (θ). The 5th input ($c[n]$) is obtained from the original sound and is represented in a different scale[3]. Due to the variety and redundancy of information, in relation to the 5 inputs, we ensure that the CGP system has a variety of representations of the same data, so that it can be able to choose the one who best fits to the problem.

4.4 Individual Encoding

Usually, Cartesian Genetic Programming contains three node types: input nodes function nodes and output nodes. Our system has 5 input nodes and only one output node. Our CGP graph contains only one row with 100 function nodes, which is a common value for this kind of approaches [12]. Each function node contains 5 different genes (see Figure 7). A function node has two genes for encoding the inputs: they can be the system inputs or the outputs of another nodes, and they are 2 because of it is the maximum arity of our function set. There is also a gene that represents the function number (F_n) from the pre-established function set. And, finally, there are two genes for the real parameters (P_1 and P_2) used as

[3]All the inputs are normalized to a max value of 1. y scale of the cepstrum graph is limited to 0.1

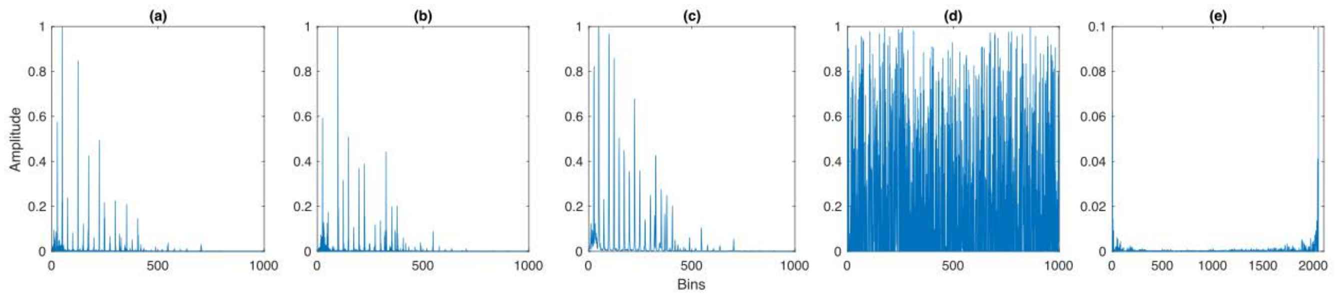

Figure 6: System Innputs, (a) real part, (b) imaginary part, (c) radius, (d) angle, (e) cepstrum

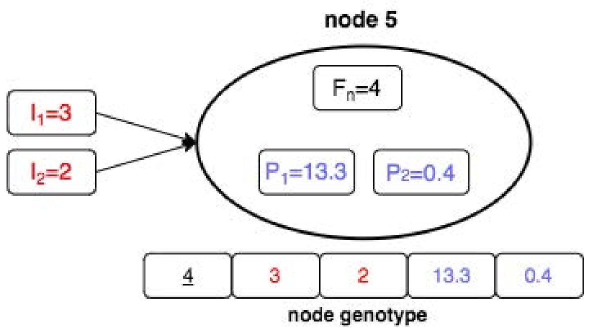

Figure 7: Function node genes (5): 2 inputs, 1 code function and 2 real parameters

parameters for the correct work of the correspondent mathematical functions from the function set.

The F_n gene represents the function used by that node chosen from the function set represented by a lookup table. Note that all the functions are prepared to receive one or two vectors and all of them return a vector. Our function set is almost comprised by filtering operations on vectors and by arithmetic operations with constants and vectors.

Figure 7 also shows that there are two real parameters in the node genotype. These parameters are useful for the functions, since each function uses at most 2 parameters to accomplish its task. In fact, most functions need real parameters. However, the same parameter has a different meaning and a different domain from function to function. To avoid a tremendous increase of parameters and genes in each node, we use one real parameter to represent constants, and other one to represent a percentage of an interval. This way, with only two additional genes, we managed to fulfill all function set needs. As the same parameter may have one meaning and domain for one particular function and another meaning and domain for another function, we normalize the domain of the parameter P_2 to a fix interval. Each function is responsible for mapping that fix interval to the correct domain for its own purpose. The parameter p_2 of each function with its own range is normalized into $[0, 1]$: all intervals are transformed from $[a, b]$ to a normalized one $[0, 1]$. By using this technique, the actual value of any parameter can be seen as a number between 0 and 1 or a percentage of the interval and the mutation process becomes standard and easier.

4.5 Mutation

Mutation process plays a fundamental role on the evolutionary process of Cartesian Genetic Programming based systems. Without crossover, mutation is the only process responsible for the generation of new individuals in an offspring. The mutation process depends of two different stages, ruled by different probability distributions functions: the first stage decides if and which gene or genes will be mutated; the second stage decides how these genes will be mutated. There is a configurable parameter, the mutation probability that represents the probability of each gene undergo a mutation. For instance, $p = 0.015$ means that each gene will mutate with a 1.5% probability. Different mutations are performed, according to the gene type and domain: if a function gene happens to be mutated, then a valid value must be chosen for selecting a new function in the function set lookup table; if a mutation occurs in a gene node input, then a valid value is the output of any previous node in the genotype or any system input; the valid values for the system output genes are the output of any node in the genotype or the address of a system input. All these mutations happen according to the discrete uniform probability distribution function for integers. Two additional genes can also mutate: the real parameters used by the functions. These parameters are important because are used by those functions to perform specific tasks. According to each function, each parameter has a specific meaning and also has its own domain range. In this case, we use the normalized interval $[0, 1]$ as mutation domain. The mutation of the real genes (function parameters) is done using the normal distribution in order to address the entire range:

$$f(x) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}}, \tag{9}$$

where $f(x)$ represents the density function of x variable, with a normal distribution. This function is also represented as $N(\mu, \sigma)$, where μ is the mean and σ is the standard deviation. To perform the mutation of a function parameter, P_{old}, we generate a new random P_{mutate} using the normal distribution $N(\mu = P_{old}, \sigma)$, with σ being configurable in our system. This way, we ensure that when a mutation occurs in a real parameter, all the parameter interval is reachable, but with higher probability to mutate to closer values.

For better performance, we keep a list of active and inactive genes and when there is no mutation in one of the active genes the fitness function is not computed.

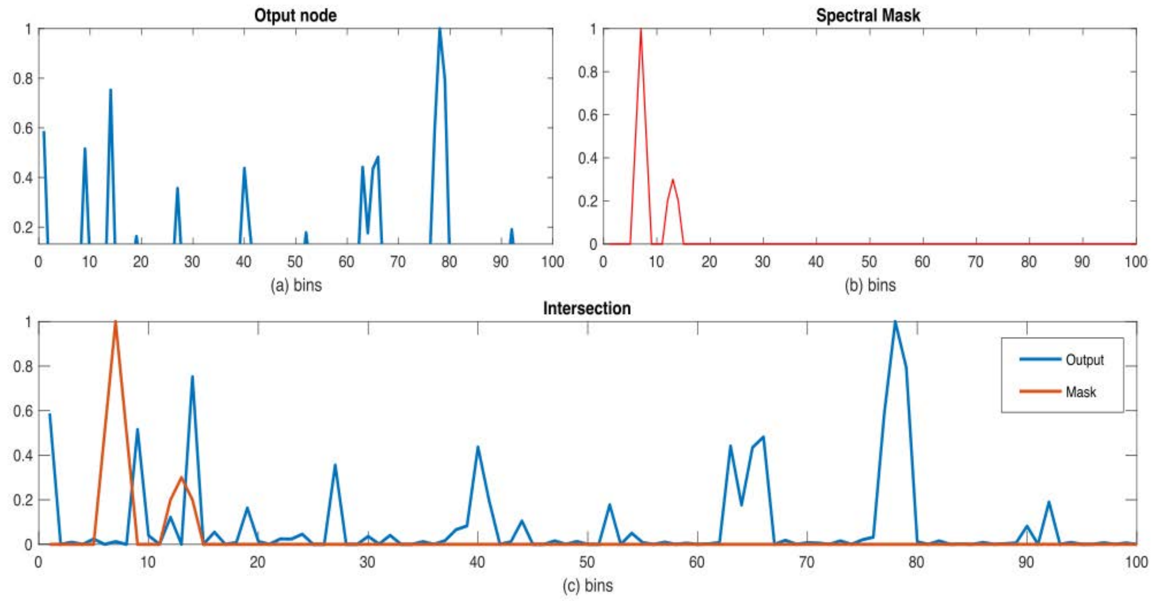

Figure 8: (a) CGP output signal, (b) harmonic mask (c) computing intersection.

Algorithm 2 Algorithm ($(1+\lambda)\ EA$)

```
1:  $t \longleftarrow 0$;
2:  Set current individual $I_0$ as the best of $\lambda$ individuals created randomly;
3:  while a stop condition is not fulfilled, do
4:      for i = 1 to $\lambda$ do
5:          Create a copy $x_i$ of current individual $I_t$;
6:          Mutate each gene of $x_i$ with probability $p$;
7:      end for
8:      Set new current individual $I_{t+1}$ as the best of $I_t \cup \{x_1, \ldots, x_\lambda\}$;
9:      $t \longleftarrow t + 1$;
10: end while
```

4.6 Evolutionary Strategy

The evolutionary strategy widely used for CGP is a special case of the $\mu + \lambda$ [4] where $\mu = 1$ (see Algorithm 2). This means that, in this special case, the population size is always $1 + \lambda$. First we select the best of λ randomly created individuals. Then, at each iteration (generation), λ new individuals are generated by applying mutations on the individual previously selected as the best among the population. Then, the best among the current individuals, 1+λ of the offspring becomes the current individual for the next iteration (generation). An offspring can become the current individual in the next iteration when it has the same fitness as the current individual and there is no other individual with a better fitness. According to Goldman [3], an empirical value for λ is 4, which was the value we used.

4.7 Binarization

Our CGP system architecture evolves one classifier for each piano note. The final output of each classifier should be a binary output: when the corresponding note is detected the output is 1, otherwise it is 0. In order to evolve each classifier there is a training stage (see Section 4.1). During this stage, each generated individual for each classifier is evaluated based on a fitness function. To compute the fitness we use F-measure, which is a measure of a test's accuracy for binary classification. Therefore, for each audio frame used as input, it is generated a binary output for each classifier. However, our CGP system is comprised of mathematical functions whose arguments are vectors and return vectors only. Thus, at the end of each CGP graph (output node), we have a vector of float values. A binarization process was added at the end of each classifier to transform its output vector into a binary output. This is done by applying a spectral mask with harmonic information: each classifier uses its own mask built by the system. The fundamental frequency of the corresponding note being detected by the classifier (F_0) is computed and 2 or more triangles are placed centered in $2^n \times F_0$ (see Figure 8 (b)), in this case we use a Harmonic Mask Parameter $n = 1$, this means that we will use 2 triangles ($n = 0$ and $n = 1$). Triangle's amplitude and width are also configurable parameters as well as the number of harmonics. To calculate the F_0 for a piano note (key) we use the following Equation that gives the fundamental frequency $F_0(n)$ of the n^{th} key.

$$F_0(n) = 440 \times 2^{\frac{n-49}{12}} Hz, \tag{10}$$

In order to accomplish a binary output, we use a comparison process between the CGP output vector normalized in amplitude $O_{cgp}(n)$ and the spectral harmonic mask with the frequency corresponding to the pitch of the estimator, $M_{F0}(n)$. The first step is the normalization of the output vector in amplitude. This way all the elements of the vector fall in the interval [0;1]. Then, we generate the following scalar computing the inner product between the 2 vectors:

$$x = \sum_{n=0}^{N} O_{cgp}[n] \times M_{F0}[n], \tag{11}$$

where x measures the discrete intersection between the two discrete signals. If we approximate these signals to a continuous domain, we could see x as the intersected area between the two signals.

Finally, we used a threshold function to accomplish the binary result:

$$T(x) = \begin{cases} 1, & \text{if } x > \theta \\ 0, & \text{if } x <= \theta \end{cases} \tag{12}$$

where θ is the threshold value. Since both signals are normalized, the max value for x is:

$$x_m = \sum_{n=0}^{N} M_{F0}[n]. \tag{13}$$

This threshold value also evolves (mutates) during the training stage. This way, besides the genes mutation also the threshold may adapt to reach a better fitness in a faster way. The threshold mutation probability is configurable, and was empirically set to the same mutation probability of the other genes. When the system decides to mutate the threshold its mutation values are ruled by the normal distribution $N(\mu = \theta_{old}, \sigma)$, which is confined to the interval: $[0; x_m]$. The new threshold depends on the old threshold, that is: the mean of the normal probability distribution that generates the new value.

4.8 Fitness evaluation

Thanks to the binarization process, each CGP generated graph can act as a classifier and dictate if the corresponding piano key is present on the input audio signal. This way, during the evolutionary training process, each classifier can be evaluated according its classification. This evaluation is done using F-measure (Equation 14).

$$F_{measure} = 2 \times \frac{recall \times precision}{recall + precision}, \tag{14}$$

where:

$$precision = \frac{tp}{tp + fp}, recall = \frac{tp}{tp + fn}. \tag{15}$$

The main goal of the training or evolutionary process is maximize the $F_{measure}$ in order to reach the most accurate as possible classifier. During the test phase the fitness function is also computed to evaluate the results of the evolved classifiers for a test data set.

5 EXPERIMENTS AND RESULTS

To make a detailed study of the proposed system results, we evolved 61 classifiers during the training stage, each one for the correspondent piano key, from C2 (MIDI note 36) to C7 (MIDI note 96). Each piano key is represented by the corresponding MIDI note number, being 60 the MIDI note number corresponding to the C4 musical note (the middle C).

The experiments comport two distinct stages: the training stage where all 61 classifiers are evolved and the testing stage where we test those classifiers to accomplish an accuracy value, using the F-measure metric.

Table 1: Training Parameters

Parameter	Value
Frame Size	4096
Fitness Initial Threshold	1.5
Positive Test Cases	100
Negative Test Cases	100
Outputs	1
Rows	1
Columns	100
Levels Back	100
λ *(E.S. 1+λ)*	4
Mutation Probability	5%
Threshold Mutation Probability	6%
Harmonic Mask	1
Runs	30
Generations	10000

5.1 Training

The training data set was extracted from MAPS database [2] and it is composed of both monophonic and polyphonic sounds, being the polyphonic extracted from the subset called random chords subset. These are chords composed by random notes without any music rules, like harmonicity and musical consonance. Training the classifiers with random chords is an important characteristic because this way, the evolved classifiers will not be constrained to any kind of music, like for example western music neither constrained to any music rules.

The configurable parameters of our proposed CGP system used in our experiments are presented in Table 1. Also, for the individuals encoding, we used on single row with 100 nodes. The "Harmonic Mask" value was empirically set to 1: the CGP system is prepared use harmonic mask since 1st harmonic to the 5th. The evolutionary process consisted of 30 runs with 10000 generations each, using 100 positive and 100 negative cases. The classifiers were evaluated using F-measure metric, explained in Equation 14. The training results are illustrated in Figure 9: the F-measure mean result for the 61 classifiers reached 95,1%.

5.2 Test

The training results give us an idea about the quality of the classifiers however the important results are those obtained in the test stage. During the test phase we use the obtained classifiers to identify the presence of their corresponding notes or piano keys in monophonic and polyphonic audio files. We measure the quality of the classifiers by testing them with a different data set: the test set. The test set for each classifier consisted in 800 piano audio files of chords and single notes obtained from the MAPS database [2]. The training set and test set are disjointed. Figure 10 shows the graph with the achieved performance. During our test stage the F-measure obtained is up to 73%.

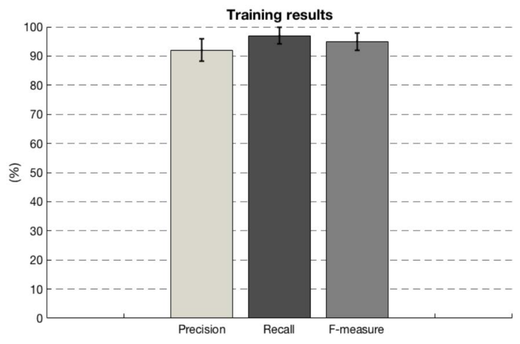

Figure 9: Training results for 61 Classifiers.

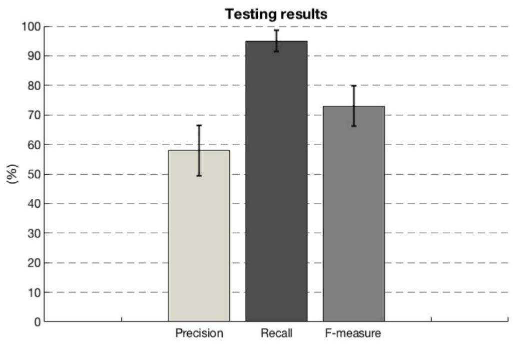

Figure 10: Test results for 61 Classifiers.

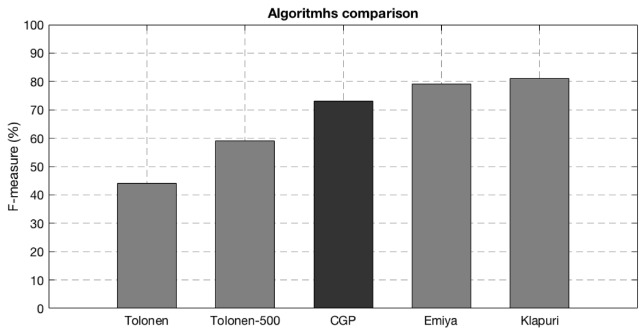

Figure 11: Comparison with the state of the art.

5.3 Comparison with the state of the art

Although there is a lack of Evolutionary approaches on multi-pitch estimation problems in the literature, specially with multiple independent classifiers architecture, we compare our results with 4 different state of the art algorithms for multi pitch estimation [1].

The overall results shown in Figure 11 demonstrate that our system reaches 73% in F-measure, being the third best in quality, only behind Emiya with 79% and Klapuri with 81%. This is also the only one using an Evolutionary Algorithm (CGP) and with a parallel architecture with independent classifiers. Our F-measure results are in line with the state of the art algorithms, and with much room to improve. Unlike Klapuri and Emiya algorithms, our architecture is highly parallelizable: since the evolved classifiers are independent, they can run in parallel. Our technique has been preliminary tested on a 8 core processor and, due to its parallelization capabilities, it worked on real time.

6 CONCLUSIONS AND FUTURE WORK

This article explains and details the Cartesian Genetic Programming strategy used to address the problem of multi pitch recognition on piano polyphonic sounds. Our F-measure results are among the state of the art for this particular task, reaching over 73% in F-measure. Another peculiar characteristic is the use of Genetic Programming as methodology. This methodology also allows us to learn, analyze and even improve the resulting classifiers, because we can analyze the final mathematical functions evolved for each classifier, decoding the individual genotype.

This EA system based on CGP generates one independent classifier for each piano note. This singularity is a great advantage in time consumption because it allows parallelization in a simple way, as illustrated in Figure 3. For one audio signal to be tested we have to run all the 61 classifiers to find which notes it contains. The system architecture allows classifiers to be run on different machines or in the same machine but using the multi-core processor architecture. This way, we can run simultaneously different note classifiers reducing the time to process an audio signal.

The independent classifiers and the parallelization process has been analyzed in a preliminary study, and important time gains have been observed. This will be fully checked in future work, but we are optimistic about the results the technique described might provide.

ACKNOWLEDGMENTS

We acknowledge support from Spanish Ministry of Economy and Competitiveness under project TIN2017-85727-C4-{2,4}-P, Regional Government of Extremadura, Department of Commerce and Economy, the European Regional Development Fund, a way to build Europe, under the project IB16035 and Junta de Extremadura, project GR15068 and project GR18049.

REFERENCES

[1] Valentin Emiya, Roland Badeau, and Bertrand David. 2010. Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle. *Audio, Speech, and Language Processing, IEEE Transactions on* 18, 6 (2010), 1643–1654.

[2] Valentin Emiya, Nancy Bertin, Bertrand David, and Roland Badeau. 2010. *MAPS - A piano database for multipitch estimation and automatic transcription of music.* Research Report. 11 pages. https://hal.inria.fr/inria-00544155

[3] Brian W Goldman and William F Punch. 2015. Analysis of cartesian genetic programming's evolutionary mechanisms. *IEEE Transactions on Evolutionary Computation* 19, 3 (2015), 359–373.

[4] Nikolaus Hansen, Dirk V. Arnold, and Anne Auger. 2015. *Evolution Strategies.* Springer Berlin Heidelberg, Berlin, Heidelberg, 871–898. https://doi.org/10.1007/978-3-662-43505-2_44

[5] Simon Harding, Jürgen Leitner, and Juergen Schmidhuber. 2013. Cartesian genetic programming for image processing. In *Genetic programming theory and practice X.* Springer, 31–44.

[6] Tiago Inácio, Rolando Miragaia, Gustavo Reis, Carlos Grilo, and Francisco Fernandéz. 2016. Cartesian genetic programming applied to pitch estimation of piano notes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI).* IEEE, 1–7.

[7] A. P. Klapuri. 2003. Multiple fundamental frequency estimation based on harmonicity and spectral smoothness. *IEEE Transactions on Speech and Audio Processing* 11, 6 (Nov 2003), 804–816. https://doi.org/10.1109/TSA.2003.815516

[8] John R Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection.* Vol. 1. MIT press.

[9] John R Koza. 1994. Genetic programming II: Automatic discovery of reusable subprograms. *Cambridge, MA, USA* (1994).

[10] M. Marolt. 2004. A connectionist approach to automatic transcription of polyphonic piano music. *IEEE Transactions on Multimedia* 6, 3 (June 2004), 439–449. https://doi.org/10.1109/TMM.2004.827507

[11] Luis Gustavo Martins. 2008. *A computational Framework for Sound Segregation Music Signals*. Ph.D. Dissertation. University of Porto, Porto, Portugal.

[12] J.F. Miller. 2011. *Cartesian Genetic Programming*. 17–33 pages. https://doi.org/10.1007/978-3-642-17310-3

[13] Julian F Miller. 2013. GECCO 2013 tutorial: cartesian genetic programming. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. ACM, 715–740.

[14] Julian Francis Miller and Simon L Harding. 2008. Cartesian genetic programming. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*. ACM, 2701–2726.

[15] Rolando Miragaia, Gustavo Reis, Francisco Fernandéz, Tiago Inácio, and Carlos Grilo. 2018. CGP4Matlab-A Cartesian Genetic Programming MATLAB Toolbox for Audio and Image Processing. In *International Conference on the Applications of Evolutionary Computation*. Springer, 455–471.

[16] M. Mueller and F. Wiering (Eds.). 2015. *An efficient temporally-constrained probabilistic model for multiple-instrument music transcription*. ISMIR, Malaga, Spain.

[17] A Michael Noll and Manfred R Schroeder. 1964. Short-Time "Cepstrum" Pitch Detection. *The Journal of the Acoustical Society of America* 36, 5 (1964), 1030–1030.

[18] Alan V Oppenheim and Ronald W Schafer. 2004. From frequency to quefrency: A history of the cepstrum. *IEEE signal processing Magazine* 21, 5 (2004), 95–106.

[19] G Reis, F Fernandéz de Vega, and A Ferreira. 2012. Audio Analysis and Synthesis-Automatic Transcription of Polyphonic Piano Music Using Genetic Algorithms, Adaptive Spectral Envelope Modeling, and Dynamic Noise Level Estimation. *IEEE Transactions on Audio Speech and LanguageProcessing* 20, 8 (2012), 2313.

[20] Gustavo Reis, Nuno Fonseca, Francisco Fernandez, and Aníbal Ferreira. 2008. A genetic algorithm approach with harmonic structure evolution for polyphonic music transcription. In *Signal Processing and Information Technology, 2008. ISSPIT 2008. IEEE International Symposium on*. IEEE, 491–496.

[21] Gustavo Reis, Nuno Fonseca, Francisco Fernández de Vega, and Anibal Ferreira. 2008. Hybrid genetic algorithm based on gene fragment competition for polyphonic music transcription. *Applications of Evolutionary Computing* (2008), 305–314.

[22] Chunghsin Yeh, Axel Roebel, and Xavier Rodet. 2010. Multiple Fundamental Frequency Estimation and Polyphony Inference of Polyphonic Music Signals. *Trans. Audio, Speech and Lang. Proc.* 18, 6 (Aug. 2010), 1116–1126. https://doi.org/10.1109/TASL.2009.2030006