Cartesian genetic programming applied to pitch estimation of piano notes

Tiago Inácio*, Rolando Miragaia*, Gustavo Reis*, Carlos Grilo* and Francisco Fernandéz†
*School of Technology and Management
Polytechnic Institute of Leiria, Portugal
Email: firstname.lastname@ipleiria.pt
†University of Extremadura
Mérida, Spain
Email: fcofdez@unex.es

***Abstract*—Pitch Estimation, also known as Fundamental Frequency (F0) estimation, has been a popular research topic for many years, and is still investigated nowadays. This paper presents a novel approach to the problem of Pitch Estimation, using Cartesian Genetic Programming (CGP). We take advantage of evolutionary algorithms, in particular CGP, to evolve mathematical functions that act as classifiers. These classifiers are used to identify piano notes' pitches in an audio signal. For a first approach, the obtained results are very promising: our error rate outperforms two of three state-of-the-art pitch estimators.**

1. Introduction

Pitch estimation on sound signals, also known as F0 detection, is a very important task of Automatic Music Transcription. This is a process in which a computer program writes the instrument's partitures of a given song or an audio signal. Usually, only pitched musical instruments are considered. Music transcription is a very difficult problem from both musical and computational points of view: although there has been much research devoted to it, it still remains an unsolved problem.

Given that Cartesian Genetic Programming (CGP) has already demonstrated its capabilities for synthesizing complex functions capable of extracting main features from images and performing image segmentation [1], we wanted to test its capabilities, when applied to audio processing, specially on pitch estimation. To tackle the problem of Pitch Estimation by using CGP, and future problems related to audio signal processing, we created a CGP toolbox for Matlab. CGP toolbox stands for *Cartesian Gentic Programming for Sound Processing*. Then, by using this toolbox, we created a CGP system to synthesize mathematical expressions which act as classifiers capable of identifying the pitch of all piano keys.

The rest of this section explains the related work. Section 2 presents the Cartesian Genetic Programming features, Section 3 explains our approach and on Section 4 we show our experiments and results. Finally on Section 5 we present our conclusions and future work.

1.1. Related Work

Over the years, there has been a lot of research on Pitch Estimation. However, to the best of our knowledge, there are no Cartesian Genetic Programming approaches for addressing this problem. Yeh et al. [2] proposed an algorithm based on the short-time Fourier transform (STFT) representation, by applying an adaptive noise level estimation algorithm and an harmonic matching technique. Klapuri [3], proposed an iterative approach algorithm, based on harmonicity and spectral smoothness. Reis et al. [4], used a genetic algorithm approach which relies on an adaptive spectral envelope modeling and dynamic noise level estimation. Marolt [5], presented a connectionist approach where he uses a partial tracking technique, based on an auditory model, which converts the acoustic signal into time-frequency space, and uses adaptive oscillators to detect periodicities in the output of the auditory model. Benetos and Weyde [6], 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.

2. Cartesian Genetic Programming

Evolutionary algorithms encode each possible solution (called an individual) to the problem as a set of genes. During the evolutionary process, the genes of each individual, or possible solution to the problem, are mutated and possibly recombined, to create better and fitter individuals. At the end of each iteration (generation) all the individuals are evaluated and, according to their quality (fitness value) as a solution to the problem, some pass to the next generation and some are discarded.

Cartesian Genetic Programming is an increasingly popular and efficient form of Genetic Programming [7], [8] proposed by Julian Miller in 2000 [9]. In its classic form, it uses a very simple integer based genetic representation of a program in the form of a directed graph.

The genotype is a list of integers (and possibly real parameters) that represent the program primitives and how they are connected together (see Figure 1). The programs

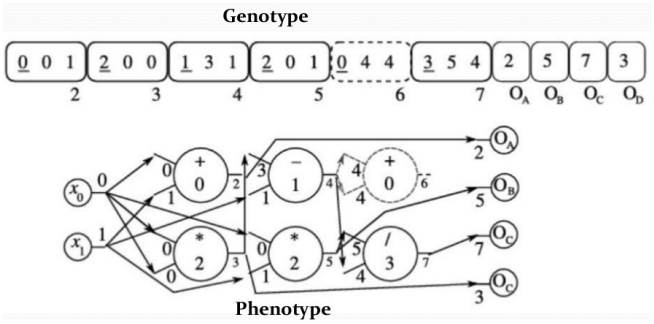

Figure 1. General form of a CGP graph (program). It is a grid of nodes whose functions are chosen from a set of primitive functions. There are 2 inputs and 3 outputs. The grid has $n_c = 3$ (columns) and $n_r = 2$ (rows).

are represented as graphs in which there are non-coding genes. The genes are addresses in data (connection genes), addresses in a look up table of functions and additional parameters. This representation is very simple, flexible and convenient for many problems. Figure 1 shows the general form of a CGP graph. Usually, all functions have as many inputs as the maximum function arity and unused connections are ignored.

CGP is *Cartesian* because it considers a grid of nodes that are addressed in a *Cartesian* coordinate system. Each CGP graph node may contain additional genes for encoding additional parameters that might be necessary for specific functions (eg.: a threshold value). CGP uses a population of individuals, being each one a candidate solution to the problem to be solved. A fitness function is used to quantitatively evaluate individuals on each iteration (generation).

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: **end while**

CGP algorithm shown in Algorithm 1, begins with the generation of the initial population, then it uses a fitness function to evaluate the individuals of the population. The evolutionary strategy chooses the fittest one (best individual) and promote it directly the next generation. The remaining places in the population are mutated versions of the fittest individual. The algorithm stops when the stopping criterion is reached.

3. CGP approach to Pitch Estimation

CGP in its general form is a grid of nodes whose functions are chosen from a set of primitive functions. The grid has n_c (columns), n_r (rows) and $levels\text{-}back$ which determines how many previous columns of cells may have their outputs connected to a node in the current column. The idea is illustrated in Figure 1. Depending on n_r, n_c and

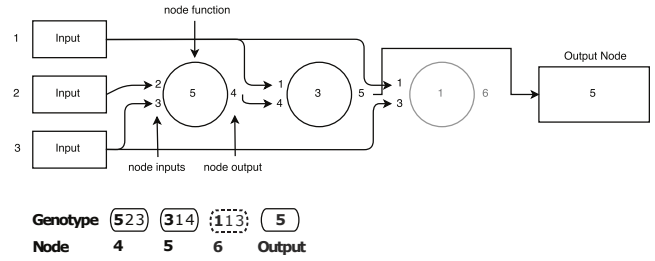

Figure 2. CGP graph with multiple inputs, one row, one output and its resulting genotype.

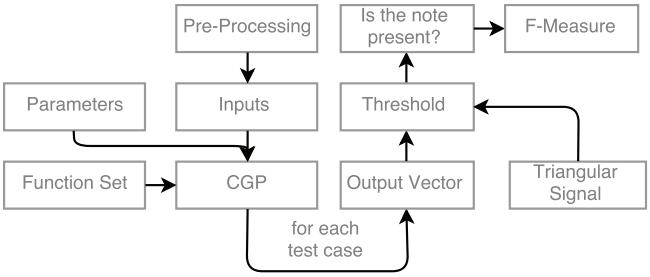

Figure 3. System architecture

$levels\text{-}back$, a wide range of graphs can be generated, when $n_r = 1$ and $levels\text{-}back = n_c$, arbitrary directed graphs can be created with a maximum depth. In general, choosing these parameters imposes the least constraints, so, without specialist knowledge, this is the best and most general choice [10]. In our particular CGP approach, we have multiple inputs, only one row of graph nodes, one output (the result of the corresponding classifier), and $levels\text{-}back = n_c$ as depicted in Figure 2. The resulting graph and genotype codification is also shown.

To perform the pitch detection using CGP, we developed a system where some important decisions and tasks were made besides the CGP. We defined what kind of inputs to use from the original piano audio signal, through a preprocessing task. We also had to develop a process to reach a binary output in order to perform our fitness function. The block diagram of our proposed system is much more than a simple CGP process and is depicted in Figure 3. Each step of this system is described carefully throughout this section.

3.1. Preprocessing

The first task of the proposed system is the Preprocessing. For this, we used the piano samples (audio signals) from MAPS database [11]. This is a huge dataset with multiple piano melodies in wave format. The piano sound signals are vectors in time with a sample rate of 44.100 samples per second.

For the preprocessing task, we split each piano sound signal into frames of 4096 samples width, corresponding to 96 milliseconds. To accomplish a good base for signal processing common tasks, we transform the domain signals

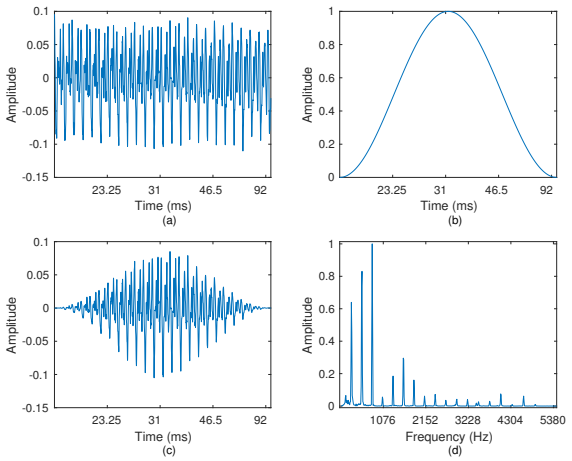

Figure 4. Preprocessing process: (a) input time signal piano note, (b) Hanning window, (c) resulting windowed signal, (d) frequency domain signal

from the time domain to the frequency domain, using the Discrete Fourier Transform (DFT), shown in Equation 1. In order to do so, each frame is windowed (see Equation 2) using an Hanning window (see Equation 3) to avoid spectral leakage. Then, the DFT is applied, obtaining signal frames in the frequency domain.

$$X[k] = \sum_{n=0}^{N-1} x_w[n] e^{-j\left(\frac{2\pi}{N}\right)nk}, (k = 0, 1, ..., N-1). \quad (1)$$

$$x_w[n] = w[n].x[n]. \quad (2)$$

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

Typically, the DFT is computed using the Fast Fourier Transform algorithm (FFT), because it is faster, the difference in speed can be enormous. The resulting frequency domain signal used in our system is obtained using Equation 1, where $x_w[n]$ is the time signal windowed and N is the number of samples in the vector. Recall that any 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. Above $\frac{f_s}{2}$ all signal energy is reflected back into the frequency range $-\frac{f_s}{2}$. Between $\frac{f_s}{2}$ and f_s, the reflection is in reverse order, which gives rise to a DFT frequency domain period $[0; f_s]$.

Due to the DFT spectrum properties, the symmetry of the real part and the antisymmetry of the imaginary part relative to Nyquist frequency, $\frac{f_s}{2}$, we may only use half of the resulting signal of the DFT, so only the first 2048 frequency bins were considered. The preprocessing process is illustrated in Figure 4, from the input time signal of a piano note to generated CGP input in frequency domain.

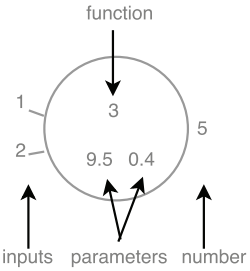

Figure 5. Node Genes(5): inputs, code function and real parameters

3.2. Individual Encoding

In general, when working with CGP, the genotype is composed by input nodes, function nodes and the output node. Our proposed algorithm contains 4 input nodes and one single output node. We used a single row CGP configuration with 100 function nodes. Each function node contains 5 genes (see Figure 5): two inputs, corresponding to the maximum function arity (the number of arguments or operands that the function takes), one integer corresponding to the function used from the function set and two function parameters. All function nodes return a vector. However, since our system is a classifier, the final output must return a binary value. In order to transform the output vector into a binary value, our system uses a threshold after a comparison between the output and a base signal. This base signal is a triangular signal centered on frequency bin of the corresponding Fundamental Frequency of the note classifier. This threshold function has a constraint to ensure it is the last function before the binary output. This threshold is also evolved: the parameter θ mutates from an initial configurable value with an also configurable mutation probability.

3.2.1. Inputs. The algorithm inputs are obtained from the preprocessing system (see Subsection 3.1). Each piano sample is split into time frames and transformed into frequency domain using a DFT. By doing so, we get frames with 2048 samples with complex domain numbers, each one representing a time frame of a piano sample note with 96 milliseconds duration. From this vector $X[k]$ in the frequency domain we may use two different representations of a complex number, Cartesian and polar:

1) $\Re\{X[k]\}$
2) $\Im\{X[k]\}$
3) $\measuredangle\{X[k]\}$
4) $||X[k]||$

This way, we have a pair of vectors each one with 2 components, making 4 usable inputs. By having redundant information, regarding the 4 inputs, we ensure the CGP system has a variety of representations of the same data, so it can be able to choose the one which best fits the problem.

3.2.2. Function set. As depicted in Figure 5, each function node has a gene that indicates which function of the function set should be used on that node. Table 1 shows our current

function set or look-up table. Note that all the functions are prepared to receive one or two vectors and all of them return a vector; the maximum function arity is 2.

TABLE 1. FUNCTION SET LOOKUP TABLE

Index	Function	Description
1	SPAbs	Absolute value
2	SPBPGaussFilter	Band pass Gaussian filter
3	SPConvolution	Convolution
4	SPCos	Cosine
5	SPDivide	Point to point Division
6	SPFFT	Absolute value of the DFT
7	SPGaussfilter	Gaussian filter
8	SPHighPassFilter	High pass filter
9	SPIFFT	Absolute value of Inverst DFT
10	SPLog	Natural logarithm
11	SPLog10	Common logarithm
12	SPLowPassFilter	Low pass filter
13	SPMedFilter	Median filter
14	SPMod	Remainder after division
15	SPMulConst	Multiplication by constant
16	SPNormalizeMax	Normalization maximum
17	SPNormalizeSum	Normalization sum
18	SPPeaks	Find peaks
19	SPSin	Sine
20	SPSubtract	Subtraction
21	SPSum	Sum
22	SPSumConst	Sum with a constant
23	SPThreshold	Tresholding
24	SPTimes	Multiplication

Our function set is basically composed by filtering operations on vectors and by arithmetic operations with constants and vectors. Each function may also have one or two real parameters that are encoded as parameter genes, these parameters also evolve during the training process.

3.2.3. Function parameters. As previously mentioned, the functions of the instruction set may have up to two parameters. In fact, most functions need real parameters to perform their tasks. In order to evolve those parameters as well, we encoded them in the genotype as genes, so they can also mutate. We used two real parameters for each function node. This way, each function has its particular parameters with a particular meaning. Each parameter r_1, r_2 of each function has its own range. Those intervals are 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 bee seen as a number between 0 and 1 or a percentage of the interval.

3.3. Mutation

In Cartesian Genetic Programming, mutation plays a major role on the evolution process. Here, each gene may be subject to mutation according to a configurable parameter: the mutation probability. This parameter represents the probability of each gene to be mutated. For instance, $p = 0.01$ means that each gene will mutate with a 1% probability. In our case, different mutation processes are used 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 uniform probability distribution function for integers. Two additional genes can also mutate: the real parameters used by the functions of the function set. These are important parameters 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 take any value in the normalized interval $[0, 1]$ and transform it into the real interval $[a, b]$. The mutation of the real genes (function parameters) is done using the normal distribution in order to address the entire range:

$$f(x) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}}, \tag{4}$$

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, r_{old}, we generate a new random r_{mutate} using the normal distribution $N(\mu = r_{old}, \sigma)$, 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.

As previously described in Section 3.2, the output of the system is obtained by using a threshold value in order to have a binary output. This threshold mutates independently from the genotype genes, with a different configurable probability.

3.4. Evolutionary Strategy

Our evolutionary strategy is a variant of a simple evolutionary algorithm known as $1+\lambda$ [12], which is widely used for CGP: the new offspring is obtained promoting the fittest individual and generate λ new individuals trough mutation. Also, an offspring can replace a parent when it has the same fitness as its parent and there is no other population member with a better fitness (see Algorithm 2). According to Goldman [13], an empirical value for λ is 4, which was the value we used.

During the evolutionary process, there is a reasonable percentage of inactive genes. Such inactive genes have a neutral effect on the genotype fitness [14]. However, the influence of neutrality in CGP has been investigated in detail by Vassilev and Miller [15] and was shown to be extremely beneficial to the efficiency of the evolutionary process. For better computing performance, we also took in account the similarity between individuals: when an individual has the same active genes than the offspring parent, there is no need to compute its fitness.

Algorithm 2 Algorithm $((\mu + \lambda)\, EA)$

1: $t \leftarrow 0$;
2: Initialize P_0 with μ individuals chosen uniformly at random;
3: **while** a stop condition is not fulfilled, **do**
4: **for** i = 1 to λ **do**
5: choose $x_i \in P_t$ uniformly at random;
6: mutate each gene x_i with probability p;
7: **end for**
8: Create the new population P_{t+1} by choosing the best μ individuals out of $P_t \cup \{x_1, \ldots, x_\lambda\}$;
9: $t \leftarrow t + 1$;
10: **end while**

3.5. Fitness Function

The main goal of the proposed system is to evolve a classifier for each piano note. The output of each classifier is binary: when the corresponding note is detected the output is 1, otherwise it is 0. During the evolutionary process, we used as inputs an amount of piano samples with the desired pitch (fundamental frequency) and the same amount of piano samples with different pitches. Thus, for each classifier, we used 50 true positive piano samples and 50 piano samples with different pitches. During the evolutionary process, the evaluation of each individual (classifier) is done using F-measure (Equation 5).

$$F_{measure} = 2 \times \frac{recall \times precision}{recall + precision}, \tag{5}$$

where:

$$precision = \frac{tp}{tp + fp},\ recall = \frac{tp}{tp + fn}. \tag{6}$$

One of our system peculiar characteristics is the binaritazion process, since the CGP output is a signal vector processed and filtered. In order to accomplish a binary output, where 1 means the presence of the corresponding pitch in the analyzed frame and 0 means the opposite, we used a comparison process between the CGP ouput vector normalized in amplitude $O_{CGP}(n)$ and a base signal with the frequency corresponding to the pitch of the estimator, $B_{F0}(n)$. 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]. The base signal is obtained with a triangular mask on frequency domain around the F0 of the estimator. We used a triangular mask with 3 configurable points in both size and amplitude. We then generate the following scalar:

$$x = \sum_{n=0}^{N} O_{CGP}(n) * B_{F0}(n), \tag{7}$$

where x measures the interception between the two discrete time signals. If we approximate these signals to continuous time we could see x as the intersected area between the two signals. Finally, we used a threshold function to accomplish the binary result:

$$T(x) = \begin{cases} 1, & \text{if } x > \theta \\ 0, & \text{if } x <= \theta \end{cases} \tag{8}$$

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

$$x = \sum_{n=0}^{N} B_{F0}(n). \tag{9}$$

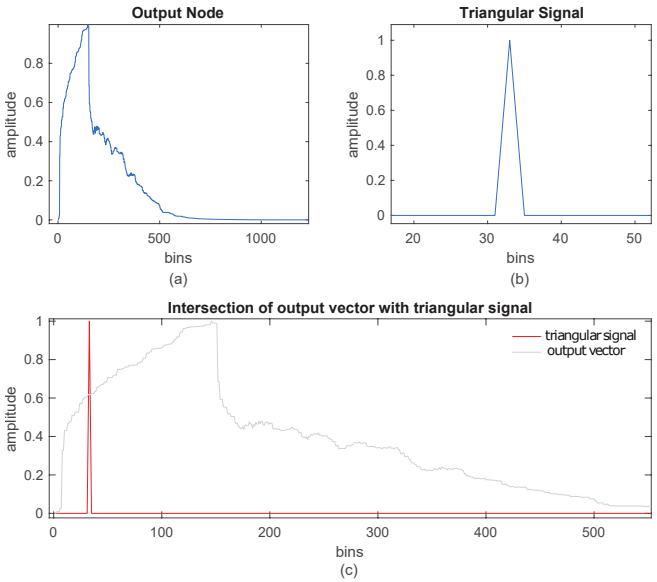

Figure 6. (a) CGP output signal, (b) base triangular signal (c) computing intersection for threshold.

All the fitness process including the sum and the thresholding is illustrated in Figure 6.

4. Experiments and Results

Since this is our first approach to the problem of Pitch Estimation of piano notes using Cartesian Genetic Programming, our main goal is to show the applicability of CGP on this problem. We evolved one classifier for each piano note. 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 piano sounds used for training and testing are those from the MAPS database [11]. Table 2 shows the values of the configurable parameters for our system. The evolutionary process consisted of 30 runs with 5000 generations each, using 50 positive and 50 negative cases. The classifiers were evaluated using F-measure (Equation 5).

Figure 7 shows the individual results of each run (i.e. evolved pitch estimator) for classifying the note 60. The best generated classifier was chosen as the pitch estimator for note 60. Figure 8 depicts the resulting program of this

TABLE 2. EXPERIMENTS PARAMETERS

Parameter	Value
Frame Size	4096
Fitness Threshold	0.5
Positive Test Cases	50
Negative Test Cases	50
Outputs	1
Rows	1
Columns	100
Levels Back	100
Population Size	4
Mutation Probability	5%
Runs	30
Generations	5000

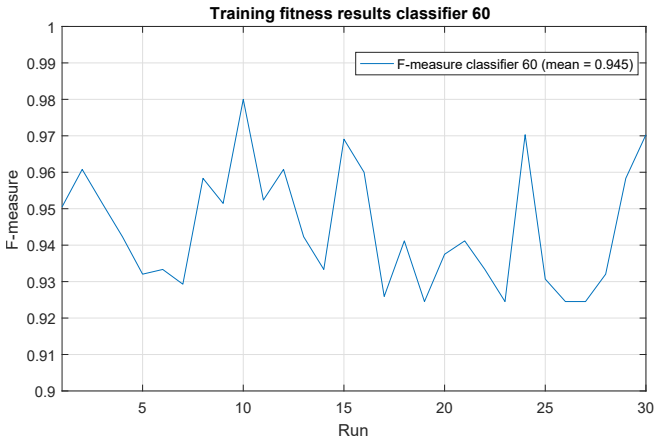

Figure 7. Training results obtained during 30 runs for pitch 60. Fitness values using F-measure

classifier. It shows the functions used in each node as well as the inputs of each node. For instance: the node 101 contains the Gaussian filter function and the function's arguments (inputs) are the outputs of the nodes 36 and 87. The resulting program is a set of mathematical functions over vectors - our phenotype.

After the training process, each classifier was tested with a different test set. Each test set consisted in 144 negative notes (48×3) and 5 positive notes, comprising a total of 149 piano sound samples. Table 3 shows our results. As a first approach, these results are very encouraging, since for almost all notes we achieved a classifier with F-Measure values greater than 70%.

In order to compare our results with the ones by other researchers we generated the graph depicted in Figure 9, where besides F-measure we can see the error rate in percentage, for the data-set test, with 96ms frames. According to Emiya [16], the three main monophonic pitch estimators are: Parametric F0 estimator, the Nonparametric F0 estimator and the YIN estimator [17] and those estimators have mean error rates of 2.4%, 3.0% and 11.0% respectively. Our CGP approach to F0 estimation reaches the mean error rate of 2.5%, a very encouraging result.

```
Node 1 = input (1)
Node 2 = input (2)
Node 3 = input (3)
Node 4 = input (4)
Node 5 = SPConvolution ( 1 , 1 )
Node 6 = SPFFT ( 1 , 4 )
Node 7 = SPConvolution ( 4 , 4 )
Node 8 = SPTimes ( 5 , 7 )
Node 10 = SPSum ( 2 , 1 )
Node 11 = SPIFFT ( 10 , 4 )
Node 12 = SPPeaks ( 8 , 11 )
Node 13 = SPPeaks ( 11 , 10 )
Node 14 = SPSum ( 3 , 10 )
Node 15 = SPSubtract ( 1 , 13 )
Node 16 = SPAbs ( 13 , 12 )
Node 17 = SPLog10 ( 16 , 10 )
Node 18 = SPThreshold ( 5 , 16 )
Node 19 = SPCos ( 17 , 18 )
Node 20 = SPLog ( 8 , 2 )
Node 23 = SPNormalizeSum ( 6 , 14 )
Node 24 = SPAbs ( 13 , 13 )
Node 33 = SPDivide ( 12 , 23 )
Node 36 = SPHighPassFilter ( 19 , 20 )
Node 87 = SPSum ( 15 , 33 )
Node 101 = SPGaussfilter ( 36 , 87 )
Node 104 = SPIFFT ( 24 , 101 )
```

Figure 8. Evolved classifier code for pitch 60

TABLE 3. TEST RESULTS FOR 19 CLASSIFIERS

classifier	tp	tn	fp	fn	f-measure
48	5	135	9	0	0.53
50	5	140	4	0	0.71
52	5	138	6	0	0.63
53	5	142	2	0	0.83
55	5	138	6	0	0.63
57	5	139	5	0	0.67
60	5	142	2	0	0.83
61	4	142	2	1	0.73
62	4	144	0	1	0.88
63	4	144	0	1	0.88
64	5	138	6	0	0.63
65	5	141	3	0	0.77
66	5	139	5	0	0.67
67	5	141	3	0	0.77
68	5	141	3	0	0.77
69	5	141	3	0	0.77
70	5	142	2	0	0.83
71	5	142	2	0	0.83
72	5	142	2	0	0.83

5. Conclusions and Future Work

This paper describes the Cartesian Genetic Programing strategy applied to pitch recognition on piano notes. This is a completely new approach to the mentioned problem: With our work and experiments, we have shown the feasibility of this technique to the problem of pitch estimation. The results are encouraging and it can bee considered a good starting point. Without a complex parameters tuning process we obtained a 2.5% error rate for our experiments with a standard sound database. The results accomplished with the CGP technique are in line or even better than the most popular algorithms for pitch recognition on piano notes.

Planning ahead, we aim to test and tune all the CGP

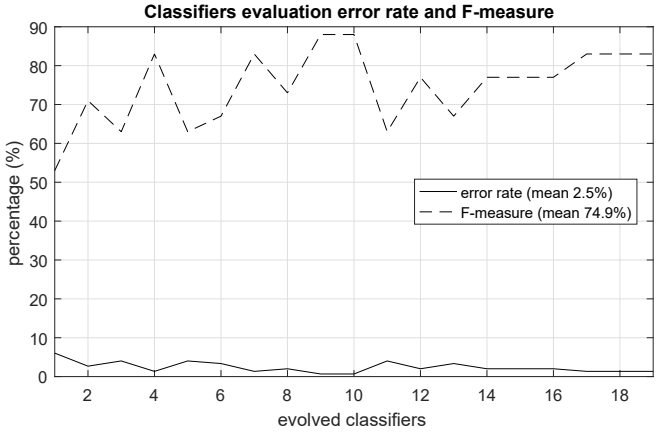

Figure 9. Graph with 19 classifiers' evaluation results in error rate and F-measure

parameters in order to obtain even better results. We are also considering taking into account additional inputs for our system, such as the generated Cepstrum of each audio frame. Another important aspect that we are planning to take into account is the use of the harmonic information during the comparison process.

References

[1] S. Harding, J. Leitner, and J. Schmidhuber, "Cartesian genetic programming for image processing," in *Genetic Programming Theory and Practice X*. Springer, 2013, pp. 31–44.

[2] C. Yeh, A. Roebel, and X. Rodet, "Multiple fundamental frequency estimation and polyphony inference of polyphonic music signals," *Trans. Audio, Speech and Lang. Proc.*, vol. 18, no. 6, pp. 1116–1126, Aug. 2010. [Online]. Available: http://dx.doi.org/10.1109/TASL.2009.2030006

[3] A. P. Klapuri, "Multiple fundamental frequency estimation based on harmonicity and spectral smoothness," *IEEE Transactions on Speech and Audio Processing*, vol. 11, no. 6, pp. 804–816, Nov 2003.

[4] G. Reis, F. Fernandéz de Vega, and A. Ferreira, "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*, vol. 20, no. 8, p. 2313, 2012.

[5] M. Marolt, "A connectionist approach to automatic transcription of polyphonic piano music," *IEEE Transactions on Multimedia*, vol. 6, no. 3, pp. 439–449, June 2004.

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

[7] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[8] ——, "Genetic programming ii: Automatic discovery of reusable subprograms," *Cambridge, MA, USA*, 1994.

[9] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming*. Springer, 2000, pp. 121–132.

[10] J. F. Miller, "Gecco 2013 tutorial: cartesian genetic programming," in *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. ACM, 2013, pp. 715–740.

[11] V. Emiya, N. Bertin, B. David, and R. Badeau, "Maps-a piano database for multipitch estimation and automatic transcription of music," 2010.

[12] M. Eigen, *Ingo Rechenberg Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologishen Evolution*. mit einem Nachwort von Manfred Eigen, Friedrich Frommann Verlag, Struttgart-Bad Cannstatt, 1973.

[13] B. W. Goldman and W. F. Punch, "Analysis of cartesian genetic programmings evolutionary mechanisms," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 359–373, 2015.

[14] J. F. Miller and S. L. Smith, "Redundancy and computational efficiency in cartesian genetic programming," *IEEE Trans. Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, 2006.

[15] V. K. Vassilev and J. F. Miller, "The advantages of landscape neutrality in digital circuit evolution," in *Evolvable systems: from biology to hardware*. Springer, 2000, pp. 252–263.

[16] V. Emiya, B. David, and R. Badeau, "A parametric method for pitch estimation of piano tones," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, vol. 1. IEEE, 2007, pp. I–249.

[17] A. De Cheveigné and H. Kawahara, "Yin, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.