UNIVERSIDAD DE EXTREMADURA

Tesis Doctoral

EVOLUCIÓN DE SISTEMA MULTICLASSIFICADOR BASADO EN PROGRAMACIÓN GENÉTICA CARTESIANA PARA ESTIMACIÓN MULTITONO DE AUDIO DE PIANO

Rolando Lúcio Germano Miragaia

Programa de Doctorado en Tecnología Aeroespacial: Ingenierías Electromagnética, Electrómica, Informática y Mecánica

La conformidad de los directores de la tesis consta en el original en papel de esta Tesis Doctoral

Dr. Francisco Fernández de Vega **Dr. Gustavo Miguel Jorge dos Reis**

2021

Thesis for the Degree of Doctor of Philosophy
This Thesis was submitted to the University of Extremadura in accordance with the criteria necessary for the award of the Doctorate Degree.

Organization:

Universidad de Extremadura; Cáceres; España

Title:

Evolución de Sistema Multiclassificador Basado en Programación Genética Cartesiana para Estimación Multitono de Audio de Piano *(Evolving a Multi-Classifier System for Multi-Pitch Estimation of Piano Music with Cartesian Genetic Programming)*

Author:

Rolando Lúcio Germano Miragaia

Supervisors:

Francisco Fernández de Vega, (Universidad de Extremadura, España)
Gustavo Miguel Jorge dos Reis, (Instituto Politécnico de Leiria, Portugal)

© Rolando Lúcio Germano Miragaia 2021

Dr. D. Francisco Fernández de Vega, Catedrático de Arquitectura y Tecnología de los Computadores de la Universidad de Extremadura,

CERTIFICA:

que la presente memoria, titulada "Evolución de Sistema Multiclassificador Basado en Programación Genética Cartesiana para Estimación Multitono de Audio de Piano" *("Evolving a Multi-Classifier System for Multi-Pitch Estimation of Piano Music with Cartesian Genetic Programming")* ha sido realizada por D. Rolando Lúcio Germano Miragaia en el Departamento de Tecnología de los Computadores y de las Comunicaciones de la Universidad de Extremadura.

Y para que conste, y en cumplimiento de la legislación vigente, firmo la presente.

Dr. D. Francisco Fernández de Vega

Dr. D. Gustavo Miguel Jorge dos Reis, profesor titular de lo Instituto Politécnico de Leiria, Leiria, Portugal

CERTIFICA:

que la presente memoria, titulada "Evolución de Sistema Multiclassificador Basado en Programación Genética Cartesiana para Estimación Multitono de Audio de Piano" *("Evolving a Multi-Classifier System for Multi-Pitch Estimation of Piano Music with Cartesian Genetic Programming")* ha sido realizada por D. Rolando Lúcio Germano Miragaia bajo mi dirección en el Departamento de Tecnología de los Computadores y de las Comunicaciones de la Universidad de Extremadura.

Y para que conste, y en cumplimiento de la legislación vigente, firmo la presente.

Dr. D. Gustavo Miguel Jorge dos Reis

Honouring Grandpa M.M. and welcoming Son R.M.

Acknowledgements

Firstly, I would Like to express my sincere gratitude to my supervisors Prof. Francisco Fernandéz de Vega and Prof. Gustavo Reis for their support, guidance, knowledge share and valuable suggestions along all these years.

I also would like to thank to Gonçalo Inácio for his tremendous help, and for keeping always available during all these years.

I am also grateful to Prof. Francisco Chávez de la O for configuring and make available the hardware resources for my research tests. He was also very understandable when I was sharing my time among his research work and my PhD research.

I would like to thank to my Spanish colleagues during these last 3 years, Jorge Alvarado and Francisco Diaz, for sharing their Spanish experiences and helping me to adapt to a new country and a new language and another culture.

I would like to leave a special thank to my Mom and Dad, Maria and Eliseu, for all the support and understanding through these years.

My final thanks goes to my girlfriend and mother of my recently born son, she knows how hard it was, she was always understanding and encouraging even in a special period of our life.

ABSTRACT

Multi-Pitch Estimation, or multiple fundamental frequency estimation, is the process of extracting the musical notation (pitches) from a given acoustic signal. Multi-Pitch Estimation is one of the tasks that belong to Content-based Music Information Retrieval. Music Information Retrieval has been drawing more attention due to the exponential growth of digital music. Nevertheless, there is a significant gap between the high-level human perception and the low-level signal features. Some music features, like rhythm and pitch, have an important role in helping bridge this gap, given that they are more closely related to the human perception of music. Among all musical instruments, piano is one of the most popular worldwide, and one of the most complex concerning pitch variety and number of simultaneous notes. These are the main reasons that motivate us to research on Multi-Pitch Estimation of piano sounds. Multiple Pitch Estimation refers to the determination of the underlying pitches of the obtained polyphonic sound. Unlike mono-pitch estimation, Multi-Pitch Estimation has to deal with issues such as the source number ambiguity and the octave ambiguity. Therefore, multi-pitch estimation is a challenging problem. Even though there has been significant research devoted to Multi-Pitch Estimation, it still remains largely unsolved.

The problem of Multi-Pitch Estimation is addressed as classification problem, with the main objective being to find the musical notes that are present in an observed sound signal. This problem is tackled using one of the most prominent and recent methodologies of the family of Evolutionary Algorithms – Cartesian Genetic Programming. Cartesian Genetic Programming is a subclass of Genetic Programming in which, programs are encoded as graphs of nodes; basically Cartesian Genetic Programming is an Evolutionary Algorithm that uses the evolutionary process to evolve mathematical expressions and functions by employing graph theory to solve a specific problem. This methodology is called "Cartesian" because it represents a program using a two-dimensional grid of nodes. It has already proved its capability when applied to classification problems in the field of image processing and signal processing.

To the best of our knowledge, there are no Cartesian Genetic Programming approaches for addressing the Multi-Pitch Estimation problem of piano sounds or any other musical instrument. This thesis presents a novel approach to the problem of Pitch Estimation, using Cartesian Genetic Programming. We take advantage of evolutionary algorithms, in particular Cartesian Genetic Programming, to search for complex mathematical functions that act as classifiers. These classifiers are used to identify piano notes or pitches in an audio signal.

This thesis describes a research that started with the need of developing a generic toolbox for Matlab, capable of aiding users to encode problems using Cartesian Genetic Programming. Using a small step iterative approach, we tackled the problem of Multi-Pitch Estimation, starting with a first approach to the problem for piano sounds. We developed an architecture based on multiple classifiers, in which there is one classifier for each piano note. The system has undergone several improvements in order to become more accurate, flexible and faster. We developed an adapted an on set detection system to perform data augmentation for the training stage, as well as a novel binarization process to calculate the fitness named “Harmonic Mask”. Taking advantage of the multi-classifier architecture we reached the real time performance using a several core processor with the classifiers distributed. We also extended the technique used for multi-pitch Estimation of piano sounds to other musical instruments like guitar and we showed its feasibility with the supported of experiments and results.

KEYWORDS

Cartesian Genetic Programming, Multiple Pitch Estimation, Automatic Music Transcription

RESUMEN

La estimación multitonal, o estimación de frecuencias fundamentales múltiples, es el proceso de extracción de la notación musical (tonos) de una señal acústica dada. La estimación de múltiples tonos es una de las tareas que pertenecen a la recuperación de información musical basada en el contenido. La recuperación de información musical ha atraído mucha atención debido al crecimiento exponencial de la música digital. Sin embargo, existe una brecha significativa entre la percepción humana de alto nivel y las características de la señal de bajo nivel. Algunas características de la música, como el ritmo y el tono, tienen un papel importante para ayudar a salvar esta brecha, dado que están más estrechamente relacionadas con la percepción humana de la música. Entre todos los instrumentos musicales, el piano es uno de los más populares en todo el mundo y uno de los más complejos en cuanto a variedad de tonos y número de notas simultáneas. Estas son las principales razones que nos motivan a investigar sobre la Estimación Tonal Múltiple de los sonidos del piano. La estimación de la afinación múltiple se refiere a la determinación de los tonos subyacentes del sonido polifónico obtenido. A diferencia de la estimación monofónica, la estimación multitono tiene que lidiar con problemas como la ambigüedad del número de fuentes y la ambigüedad de la octava. Por lo tanto, la estimación multitonal es un problema difícil. A pesar de que se ha investigado mucho sobre la estimación multitonal, sigue sin resolverse en gran medida.

El problema de la Estimación Multitono se aborda como un problema de clasificación, siendo el objetivo principal encontrar las notas musicales que están presentes en una señal sonora observada. Este problema se aborda utilizando una de las metodologías más destacadas y recientes de la familia de los Algoritmos Evolutivos – la Programación Genética Cartesiana. La Programación Genética Cartesiana es una subclase de la Programación Genética en la que, los programas se codifican como grafos de nodos; básicamente la Programación Genética Cartesiana es un Algoritmo Evolutivo que utiliza el proceso evolutivo para evolucionar expresiones y funciones matemáticas empleando la teoría de grafos para resolver un problema específico. Esta metodología se denomina "cartesiana" porque representa un programa mediante una cuadrícula bidimensional de nodos. Ya ha demostrado su capacidad cuando se aplica a problemas de clasificación en el campo del procesamiento de imágenes y de señales.

Hasta donde sabemos, no existen enfoques de Programación Genética Cartesiana para abordar el problema de la Estimación Multitono de los sonidos del piano o de cualquier otro instrumento musical. Esta tesis presenta una aproximación novedosa al problema de la Estimación de Tonos, utilizando la Programación Genética Cartesiana. Aprovechamos los algoritmos evolutivos, en particular la Programación Genética Cartesiana, para buscar funciones matemáticas complejas que actúan como clasificadores. Estos clasificadores se utilizan para identificar las notas o tonos de un piano en una señal de audio.

Esta tesis describe una investigación que comenzó con la necesidad de desarrollar una caja de herramientas genérica para Matlab, capaz de ayudar a los usuarios a codificar problemas utilizando la Programación Genética Cartesiana. Utilizando una aproximación iterativa de pequeños pasos, abordamos el problema de la Estimación Multitono, comenzando con una primera aproximación al problema para sonidos de piano. Desarrollamos una arquitectura basada en clasificadores múltiples, en la que hay un clasificador para cada nota de piano. El sistema ha sufrido varias mejoras para ser más preciso, flexible y rápido. Hemos desarrollado un sistema adaptado de detección de conjuntos para realizar el aumento de datos para la etapa de entrenamiento, así como un novedoso proceso de binarización para calcular la aptitud denominado "Máscara armónica". Aprovechando la arquitectura de multiclasificadores alcanzamos el rendimiento en tiempo real utilizando un procesador de varios núcleos con los clasificadores distribuidos. También extendimos la técnica utilizada para la estimación multitono de sonidos de piano a otros instrumentos musicales como la guitarra y demostramos su viabilidad con el apoyo de experimentos y resultados.

PALABRAS CLAVES

Programación genética cartesiana, estimación de tonos múltiples, transcripción automática de música

Acronyms

ACF Aoutocorrelation Function. 37

AI Artificial Intelligence. 4

AMT Automatic Music Transcription. 1

ANSI American National Standard Institute. 35

CGP Cartesian Genetic Programming. 4

CGP4Matlab Cartesian Genetic Programming toolbox for Matlab. 79

dB Decibel. 8

DFT Discrete Fourier Transform. 22

DSP Digital Signal Processing. 19, 28

EA Evolutionary Algorithm. 4, 63

EC Evolutionary Computation. 63

EP Evolutionary Programming. 67

ES Evolutionary Strategy. 66

FFT Fast Fourier Transform. 26

FS Fourier Series. 22

FT Fourier Transform. 30

GA Genetic Algorithm. 66

GP Genetic Programming. 67

HM Harmonic Mask. 119

IDFT	Inverse Discrete Fourier Transform. 25
MIDI	Musical Instrument Digital interface. 14
MIR	Music Information Retrieval. 1
MPE	Multi-pitch Estimation. 1
OOP	Object-oriented programming. 81
PSD	Power Spectrum Density. 31
STFT	Short-Time Fourier Transform. 52, 122

Contents

List of Tables

List of Figures

List of Algorithms

Listings

Chapter 1

Introduction

The problem:

> *Multi-Pitch Estimation (MPE) or multiple fundamental frequency, F0 estimation*

Multiple pitch estimation is a sub-problem of Automatic Music Transcription AMT; it has been a popular research topic for many years and still is investigated nowadays.

The general term "Music Transcription" means to produce the notation of a musical piece or a sound which was previously unanotated or which the notation is unknown. This process transforms an acoustic sound signal into human readable notation (e.g. a music score). Therefore, when this task is performed by a software or a computer program without human interference, it is considered Automatic Music Transcription. Hence, Automatic Music Transcription is the process of converting acoustic music signals to some form of human readable notation, using computer algorithms. Automatic transcription of music is a challenging research task which covers several areas besides computer science. It includes artificial intelligence and machine leaning, signal processing, psychoacoustics, music rules and theory among others. AMT is also considered one task of music information retrieval (MIR), among MIR tasks are:

- Genre Classification;
- Recommender systems
- Music Mood Classification;
- Classical Composer Identification;
- Onset Detection;

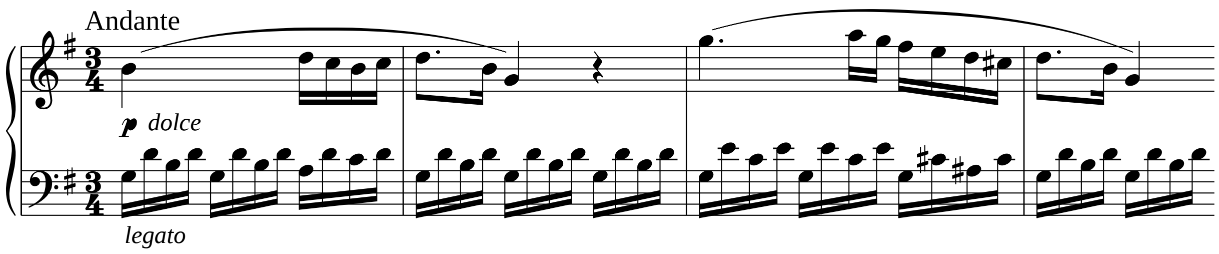


Figure 1.1: Wolfgang Amadeus Mozart's Piano Sonata No. 16 - Music score

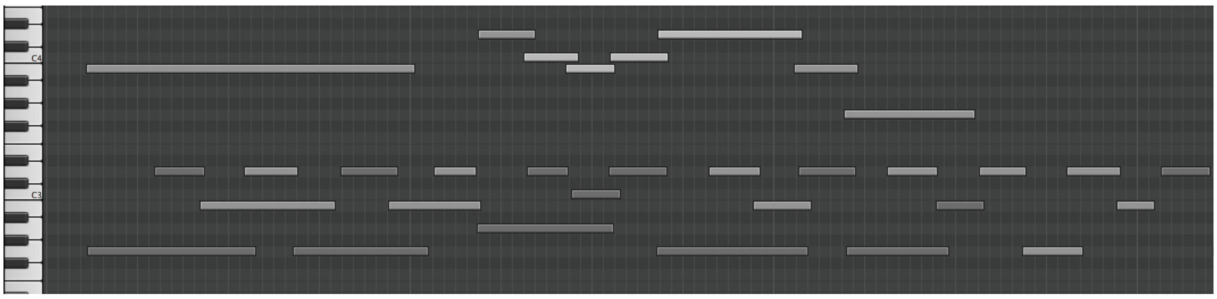


Figure 1.2: Wolfgang Amadeus Mozart's Piano Sonata No. 16 - Piano-roll.

- Real-time Audio to Score Alignment;
- Beat Tracking;
- Music generation;
- Music source separation and instrument recognition
- Music Similarity and Retrieval;
- Query by Singing/Humming;
- Melody Extraction;
- Multiple F0 Estimation;

The music transcription process, when made by human hand, is a very complex task, requiring several human capabilities and requisites, and it is only within the reach of the most talented musicians. It is also a process which consumes a large amount of time. AMT systems try to address the task of transforming an acoustic music signal into a readable document (e.g. music score) in a less time consuming way, making it more accessible for common people. Besides its main goal, AMT can be also used in other MIR tasks like plagiarism detection, music composition or artist recognition.

A musical score is a sort of guide to play a musical work and it can be represented in multiple forms. Among them, the most common is the modern notation widely used in Western tonal music, (see Figure 1.1). This representation is the desirable output for an AMT system. However, in order to extract a traditional musical score AMT leads with a set of sub problems, being the most prominent and crucial for the success of the entire system the process of estimating the multiple pitches present. Besides Multiple-Pitch Estimation, an AMT system should also estimate onset[1] time, note duration using offset, depending on the methodology used (note tracking or frame based). According to the process used, it might be also necessary to infer the meter and tonality of a music. Usually, transcription processes include two stages: the generation of a piano-roll, illustrated in Figure 1.2, from the acoustic signal of the music piece or part of it; and another stage that is responsible for transforming the piano roll into a music score. In general AMT systems focus only on generating the piano roll leaving the second stage; the conversion of a roll to a score, to other systems, considering it a separated problem.

Moreover, obtaining a score from an acoustic audio music signal might be an ambiguous task, because music is also a subjective task that depends on the performer and his mood or feelings. Therefore, musical scores may be considered general guidelines for a musician to play a musical piece. Furthermore, the process of converting a piano roll to a score, requires some high-level tasks: tempo estimation, rhythm quantization, key detection or pitch spelling. Due to these reasons, music transcription systems start with an acoustic signal and finish their job generating a piano roll. The transcription process can be applied to any musical instrument, and it can even be applied to voice, although transcription of piano music makes a bigger impact and is more complex than the majority of the common musical instruments. Pianos are classical instruments, used in almost any musical genre, being also widely used and studied. There are two basic piano configurations: grand and upright. There is also a considerable diversity in what concerns to a piano's amount of keys and sound generation process (electric, electronic, etc.). Another important feature when compared with other musical instruments is that pianos have a high level of polyphony; it is possible to press several keys at the same instant generating the same amount of notes. Taking all into account piano is considered the most challenging and complex musical instrument for music transcription.

The AMT of piano sound can be stripped of all high-level features like the metrical structure or tempo information and reduced to the fundamental task of transforming an acoustic signal into a piano roll, in which a piano roll is a simple representation of the notes present in a time instant and in a sequential form. Therefore, multiple fundamental frequency (F0) estimation is the main task and the core of the AMT process that generates the piano-roll. This main job is also denominated **Multi-Pitch Estimation**; it estimates the number of sources sounding at each time and their corresponding pitches.

The goal of Pitch Estimation is to find the pitch or fundamental frequency of a digital recording of a speech or musical note. It plays an important role, because it is the key to identify which notes are being played and at what time.

[1]Time instant in which a musical note starts.

Signals in which several sounds are played simultaneously are called polyphonic signals, in contrast to monophonic signals, in which at most one note is present at a time. Conversely, Single-Pitch Estimation identifies pitches on monophonic signals and Multi-Pitch Estimation identifies multiple pitches in polyphonic signals. Pitch Estimation of real instruments is a very hard task to address. Each instrument has its own physical characteristics, which reflects in different spectral characteristics. Furthermore, the recording conditions can vary from studio to studio and background noise must be considered.

After disclosing the problem, it is important to introduce the main methodology. Artificial Intelligence (AI) is a field of study which focuses on systems and machines that mimic human intelligence to perform some tasks; these systems are able to perceive their environment and iteratively improve themselves in a way to pursue their goal, maximizing the success chances.

All these AI features fit perfectly in the problem of Multi-Pitch Estimation. Artificial Intelligence covers many fields, like deep learning and Evolutionary Algorithms (EA). Among EAs there is a quite recent methodology, with many similarities with Genetic Programming, denominated Cartesian Genetic Programming CGP. CGP is a technique of evolving mathematical chained functions in the form of graphs, starting from a population of unfit (usually random) programs, that during the evolutionary process become fitter for a particular task. This methodology seems suitable for developing an approach to the presented problem.

1.1 Objectives and Scope of the Thesis

To the best of our knowledge, there are no Cartesian Genetic Programming (CGP) approaches for addressing the Multi-Pitch Estimation problem of pianos sounds or any other musical instrument. This thesis presents a novel approach to the problem of Pitch Estimation, using CGP. We take advantage of the EAs, and in particular CGP, to search for complex mathematical functions that act as classifiers. These classifiers are used to identify piano notes pitches in an audio signal. For example, given an audio recording of a C3 piano note, the classifier for that note should recognize that a C3 is present in that sound. There will be one classifier for each piano note. Our system's architecture is also described to show the feasibility of its parallelization and its implementation as a real-time system. Our methodology is also a white-box optimization approach that allows for the clear analysis of the solutions found and for researchers to learn and test improvements based on the new findings.

1.2 Thesis Contributions

The main contributions contained within this dissertation are summarized below:

- A CGP Toolbox for Matlab was built and it is freely available. This toolbox is generic enough to encode different problems with different requirements and it is available for public use.

- A novel approach for multi-pitch estimation of polyphonic piano sounds using CGP is presented. The results show the feasibility of the approach and validate the evolution of classifiers using CGP for Pitch Estimation.
- A multi-classifier system for multi-pitch estimation based on a distributed and parallel architecture where each evolved classifier may work independently and in parallel with the others.
- The developed approach achieves real-time using parallel capabilities of contemporaneous processors with several cores; and is also applicable for another instrument sounds, such as guitars
- The technique used for MPE of piano music is extendable to another types of musical instruments

1.3 Thesis Outline

This dissertation is organized as follows.

Chapter 2 This chapter starts by presenting a brief explanation of terminology and concepts, ranging from waves and sampling to audio signal processing. Single-Pitch Estimation approaches are presented and the problem of Multi-Pitch Estimation is also discussed.

Chapter 3 In this chapter a literature review of previous studies on multiple-F0 estimation is presented.

Chapter 4 This chapter describes the CGP technique in the context of EAs, presenting the algorithm and its main features.

Chapter 5 This chapter describes the Cartesian Genetic Programming Toolbox developed from scratch. It is shown how this toolbox can encode multiple programs, and how to configure it. An example of the application of the toolbox to a symbolic regression problem is also presented.

Chapter 6 During this chapter the methodology used to apply Cartesian Genetic Programming to the problem of Multi Pitch Estimation is presented. The developed work described is the first approach to the problem. Preliminary results are also detailed.

Chapter 7 This chapter details all the improvements made to the first approach which led to the final system version. It also presents a complete study of the obtained results and main features.

Chapter 8 The last chapter presents the main conclusions of the developed research. Suggestions to future work are also presented.

Finally, additional specific information is presented, in the form of appendices, at the end of this document: publications, details on the configuration files, examples of implementation, detailed experimental results.

Chapter 2

Signals and Sounds Background

During this chapter a brief explanation of most important background topics is presented, from waves, sound, signal sampling and more complex concepts on signal processing. This chapter was written for readers that are not contextualized with signals and systems theory. In this chapter, crucial terminology and theoretical background towards the frequency estimation of acoustic signals are presented.

2.1 Waves and Sounds

Sound is the propagation of disturbances in pressure in a medium, regardless of whether the substance of the medium is gaseous, liquid or solid, some of which can be detected by the human ear. Those disturbances are called sound waves, and propagate by repetitive variations of compression (high pressure) and rarefaction (low pressure) of the medium. The most important properties of sound waves are: wavelength (λ), frequency (f), and amplitude. The wavelength is the distance between any point in the wave and the equivalent point in the next cycle. Frequency is the number of cycles per second and it is measured in hertz (Hz).Thus, frequency is the number of times the wavelength occurs in one second. The hertz symbol (Hz) represents one cycle per second, more commonly addressed as frequency. There is a relation between frequency and wavelength and it is related with the velocity of propagation (v) of the wave (sound) in a medium:

$$v = \lambda \times f \tag{2.1}$$

The frequency range of the human ear is:

$$20Hz \leq f \leq 20kHz. \tag{2.2}$$

This means that humans can hear vibrations occurring between 20 and 20 000 times per second. Any sound with a frequency below 20 Hz known as infrasound and above than 20 kHz ultrasound.

The amplitude is the strength of a wave signal. The more amplitude the wave signal has, the more loud the volume will sound. The decibel (dB) is relative measure unit that uses a logarithmic scale:

$$dB = 10\log(\frac{I}{I_0}) \ , \quad I_0 = 10^{-12} w/m^2 \tag{2.3}$$

where I is the sound intensity measured and I_0 is the intensity of the audible threshold.

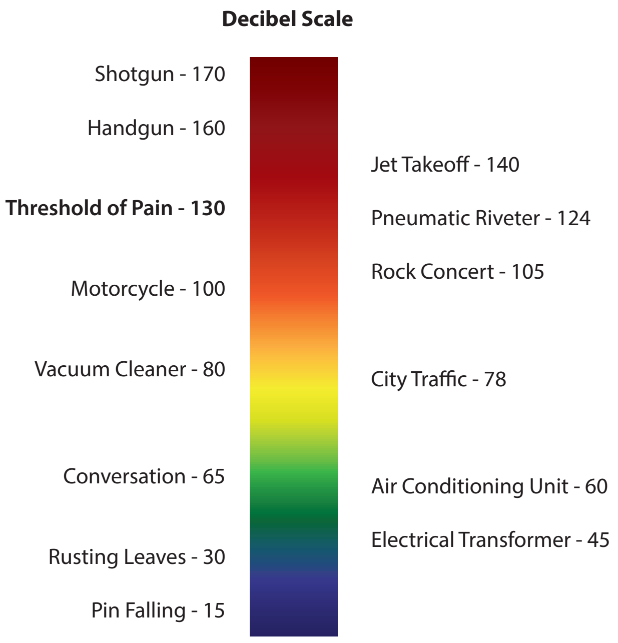


Figure 2.1: Sound intensity measured by the Decibel (dB) unit.

Among all its applications dB is used to measure sound intensity. Our ear has a logarithmic sensitivity, thus the decibel scale is commonly used to measure sound levels.

2.2 Digital Audio Signal

The process of recording and playing sound from a digital device, such as a computer, is a very complex task. It includes several stages since the sound capture until the digitization and storage.

2.2.1 AD/DA Converters

The first stage is to acquire a sound signal from the medium and transform it into an electric signal. Microphones convert acoustical energy into electrical energy, sound waves into audio signals.

The microphone essential part is a membrane that vibrates due to air pressure variations. That vibrations are converted into electric signals by an an electric circuit, nested in the microphone. The electric signal voltage varies over the time according to the membrane's vibrations. Thus, all the sound that propagates trough the air is transformed into a voltage signal due to the microphone's membrane capability of capture the air vibrations (see Figure 2.2).

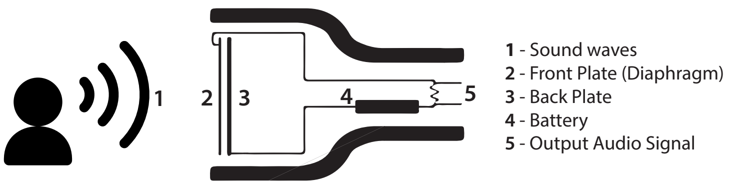


Figure 2.2: Condenser microphone scheme

The sound of an instrument reaches an acoustic-to-electric transducer (e.g. microphone) and the vibrations are converted into an electric signal which is then amplified.

The acquired electric signal is continuous in time. An analog-to-digital converter (ADC) converts the electric signal into digital data, represented by discrete numbers which are stored on a hard-drive, or any other data storage device (see Figure 2.3).

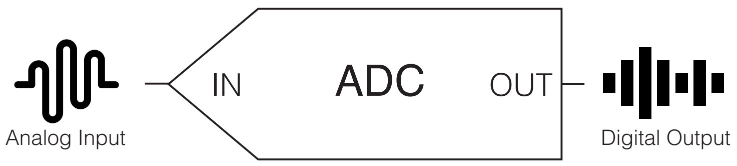


Figure 2.3: Analog-Digital converter.

To play the recorded sound, the data previously stored is transformed back to an analog signal with a digital-to-analog converter (DAC) (see Figure 2.4). The analog signal is amplified and converted to sound by an electroacoustic transducer (e.g. loudspeaker).

2.2.2 Signal Sampling

In order to convert the electrical signal into digital data, a AD converter is used. The analog-to-digital converter samples the input signal periodically in time (sampling fre-

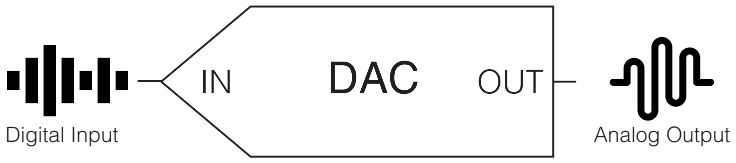


Figure 2.4: Digital-Analog converter.

quency) based on its voltage level. The voltage level is continuous in time, which means some information is lost, during the sampling process (see Figure 2.5).

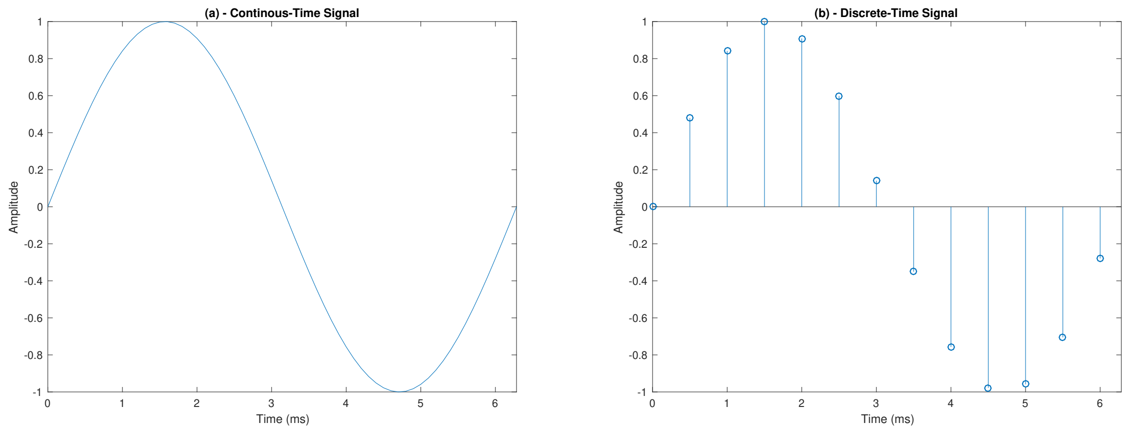


Figure 2.5: Signal sampling: **(a)** Continuous signal in time. **(b)** Sampled signal.

The AD converter uses a sampling process to transform sound oscillations into numeric values, Figure 2.5 illustrates the process with 2 graphs. This process has a frequency, i.e happens many times per second: the sampling rate. If that sequence of numbers is represented graphically it is achieved a signal wave with a similar shape to the original analog sound signal. In fact with a closer look it is possible to see that the signal is not continuous in time neither in amplitude. If the sample rate is increased (i.e. the number of samples per second) a denser graph is obtained which turns the signal more similar to the original.

In signal processing, sampling is the reduction of a continuous-time signal to a discrete-time signal. A common example is the conversion of a sound wave (a continuous signal) to a sequence of samples (a discrete-time signal), as depicted in Figure 2.6. The output of the sampler varies only in periodic intervals of time, when it assumes the instant value of the input signal.

A sample is a value or set of values at a point in time and/or space. A sampler is a subsystem or operation that extracts samples from a continuous signal. A theoretical ideal sampler produces samples equivalent to the instantaneous value of the continuous signal at the desired points. Sampling is the key concept for the real-time digital signal processing. The variations that may happen between adjacent samples are discarded and ignored (see Figure 2.7).

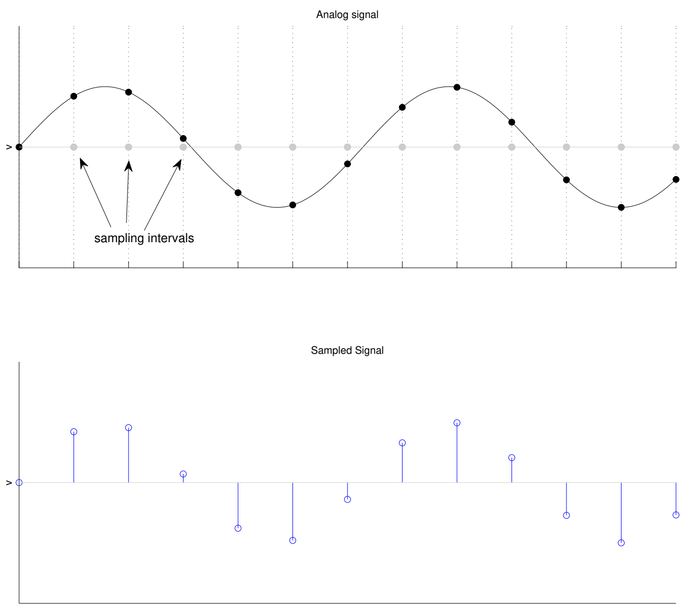


Figure 2.6: Analog signal sampled into a sampled signal

Sampling becomes the key concept for the real-time digital signal processing.

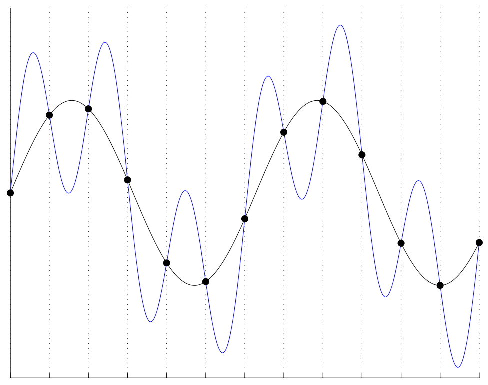


Figure 2.7: A signal composed by 2 sin waves with different frequencies, sampled with a low sampling rate, losing the higher frequency component.

Nyquist Theorem

The Nyquist theorem also known as sampling theorem, [Shannon, 1949] is as a fundamental bridge between continuous-time signals and discrete-time signals. It establishes a sufficient condition for a sample rate that permits a discrete sequence of samples to capture all the information from a continuous-time signal of finite bandwidth. The sampling theorem states: "If a low pass-band signal has the highest frequency equal to F_{max}, then for its exact reconstruction it must be sampled at least at F_s sample rate, where F_s is equal to or greater than the double of F_{max}". Equation 2.4 describes the theorem.

$$F_s \geq 2F_{max} \tag{2.4}$$

Hence, in order to a sound signal preserve its frequencies into the range from $20Hz$ to $20kHz$, the sampling rate to use should be equal or greater than $40kHz$. Therefore, the common representation is $44.1kHz$, which means 44100 samples per second, for the digital music because with those values it incorporates all the audible spectrum.

$$44.1kHz \geq 2 \times 20kHz \tag{2.5}$$

Sampling Depth

Each digital sample is a point on a 2 dimension space that can be represented in a cartesian graph were the abscissa is the discrete time and the ordinate is the signal amplitude on that precise time instant. The analog/digital converter samples the analog signal from time to time, (in the CD technologies at each $\frac{1}{44100}$ of second or $0.000023s$) and extracts the signal amplitude value and expresses it as a numeric value. However in digital medium, the possible values are also discrete and represented in a binary base. Not every values are available for representing a wave amplitude, the number of levels is finite and each level value is represented through a binary number using sequence of "bits" This way, each value of amplitude suffers an approximation to the nearest binary value, quantization.

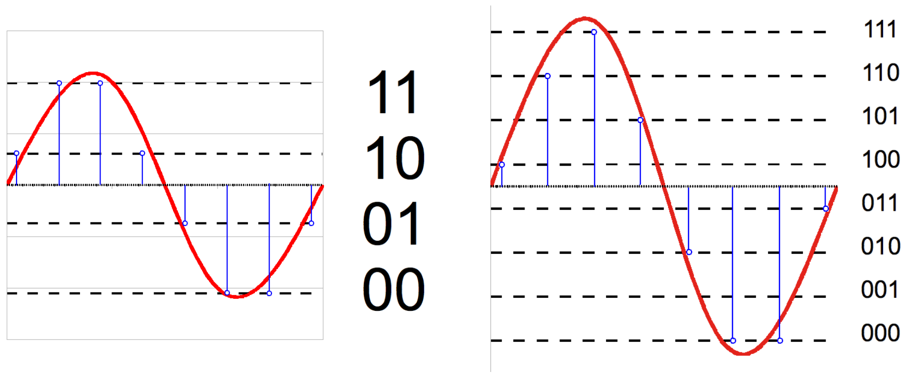


Figure 2.8: Quantization process of a signal amplitude using small bit depth. **(a)** - A 2 bit depth origins 4 levels **(b)** - A 3 bit depth origins 9 levels. Source: `https://en.wikipedia.org/wiki/Quantization_(signal_processing)`, October 2021.

Depending on the number of available bits more levels of amplitude are available to represent the sound amplitude and the difference between 2 consecutive quantization levels is smaller. Therefore, a sound sample with n bits only has 2^n possible levels. Figure 2.8 illustrates the digitalization process of a sound using 2 and 3 bit depth. An 8 bit signal representation only allow 256 amplitude levels, however the precision of the human ear is much more detailed. A human can distinguish sounds from flapping wings of a fly to an airplane's turbine, which is millions of times stronger. Using 16 bits - the common format used in audio CDs - it is possible to represent 65,536 different levels, which implies a immeasurable gain in sound precision and quality compared to the 8 bits resolution. In DVDs the amplitudes are represented by 24 bits which gives 16 777 216 distinct amplitude levels.

2.2.3 Music

There are many definitions for "Music", the Concise Oxford English [Dictionary, 2002] defines music as:

"The art of combining vocal or instrumental sounds (or both) to produce beauty of form, harmony, and expression of emotiom".

Although, according to Merriam-Webster dictionary online edition music is:

"The science or art of ordering tones or sounds in succession, in combination, and in temporal relationships to produce a composition having unity and continuity".

This definition includes two fundamental key words: *"art"* and *"science"*.

As mentioned before, sound phenomena according to physic is vibrations that travel through a medium, commonly air. The music sounds include four basic characteristics: dynamics, duration, timbre and pitch.

Dynamics
The sound dynamics is related with the perception of the sound wave amplitude. Therefore, it is physically described as the amount of energy transported in a sound wave, due to the the vibration of the particles through a medium. It is often referred to as the loudness or volume of a sound. A common dynamic indications in music, which are also referenced by their Italian words, are very soft (*pianissimo*), soft (*piano*), loud (*forte*) and very loud (*fortissimo*).

Duration
A sound produced by a musical instrument has a duration - it is audible during a period of time. Each sound has a starting instant (onset) and a ending instant (offset). The time between this two instants is known as duration. In music notation there are many symbols to represent time fractions - tempo. The tempo is the speed or pace of a music piece, it is related with the beat time (number of beats per second - bpm) or the music's rhythm.

Timbre

In music, timbre is the sound characteristic that allows us to distinguish sounds of the same frequency that were produced by different sound sources, e.g. a piano from a guitar. Humans recognize a sound of a given instrument in a music by its timbre. The physical characteristics of sound that determine the perception of timbre include frequency spectrum and envelope.

Pitch

The pitch is the **fundamental frequency - F0**, The F0 is the physical feature of the pitch. The F0 can be obtained through a physical measurement, however this may differ from the perceived pitch due to general loudness and other sound characteristics. It is a subjective attribute of sound, that is closely related to the frequency, an objective physical property. It is related to how low or how high a note sounds. Pitch is an auditory sensation that maps the vibrations of a sound wave to a tone in a musical scale.

Pitched musical instruments are generally based on a harmonic oscillator. "When an oscillator is displaced from its resting position position or equilibrium, it applies a restoring force proportional to the displacement suffered", Hooke's Law. For instance, a string instrument, when a string is plucked it suffers a restoring force proportional to the displaced relative to its equilibrium position. This physical concept causes an oscillation in the string. The string oscillation origins variations of pressure in the air and they are modeled by many frequencies. These resonant frequencies, propagate in the medium, along the string or the air and they are are self-filtered, reinforcing and also canceling each other forming standing waves. Therefore, because of the resonances physical properties including spacing, the resulting frequencies are integer multiples or harmonics of the fundamental frequency which is the lowest observable. All these multiples and the fundamental compose a series of frequecies, this is the basic ground of the harmonic series. The lowest frequency of any vibrating object is called the fundamental frequency F0. The fundamental frequency provides the sound with its strongest audible pitch reference - it is the predominant frequency in any complex waveform. It is applicable to periodic or quasi-periodic signals also representable by an harmonic series. In ambiguous situations, the period corresponding to the perceived pitch is chosen [Klapuri, 2004b].

A pitch corresponds to a a musical note (A, B, C, etc.) or more precisely to a letter and a number. An "A"played in the middle of a piano has the F0 frequency of 440 Hz. That note is represented by A440 or more usually as A4. A3 and A5 are also A's however they are one octave lower or higher respectively. According with the logarithmic scale of the octaves A4 has a F0 of A440, A3 and A5 have 220 Hz and 880 Hz, respectively. The humans pitch perception is also in a logarithmic frequency scale. This means that the distance perceived between 2 pitches A220 and A440 (separated by 220 Hz) is equal to the pitches A440 and A880, which have a difference of 440 Hz.

The pitch is commonly represented in a numeric scale based on the F0's logarithm. This way, we can adapt the MIDI standard [Association, 2008] to map the F0 f to a

Table 2.1: Pitches' values

Note	**Octave**								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
C	12	24	36	48	60	72	84	96	108
C#	13	25	37	49	61	73	85	97	109
D	14	26	38	50	62	74	86	98	110
D#	15	27	39	51	63	75	87	99	111
E	16	28	40	52	64	76	88	100	112
F	17	29	41	53	65	77	89	101	113
F#	18	30	42	54	66	78	90	102	114
G	19	31	43	55	67	79	91	103	115
G#	20	32	44	56	68	80	92	104	116
A	21	33	45	57	69	81	93	105	117
A#	22	34	46	58	70	82	94	106	118
B	23	35	47	59	71	83	95	107	119

Table 2.2: MIDI pitches' frequencies.

Note	**Octave**								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
C	16	33	65	131	262	523	1047	2093	4186
C#	17	35	69	139	278	554	1109	2218	4435
D	18	37	73	147	294	587	1175	2349	4699
D#	20	39	78	156	311	622	1245	2489	4978
E	21	41	82	165	330	659	1319	2637	5274
F	22	44	87	175	349	699	1397	2794	5588
F#	23	46	93	185	370	740	1475	2960	5920
G	25	49	98	196	392	784	1568	3136	6272
G#	26	52	104	208	415	831	1661	3322	6645
A	28	55	110	220	440	880	1760	3520	7040
A#	29	58	117	233	466	932	1865	3729	7459
B	31	62	124	247	494	988	1976	3951	7902

integer value p (see Equation 2.6).

$$p = 12 \times \log_2 \left(\frac{f}{440Hz} \right) + 69 \tag{2.6}$$

This way we may consider a linear distribution for pitches, where each octave contains 12 notes (piano keys) and each note differs one semitone. Taking as example the note A4, it has the F0 frequency 440 Hz and corresponds to pitch number 69. The numeric scale of pitches is shown in Table 2.1 and Table 2.2 presents the correspondence in frequencies where C0 frequency $\approx 16Hz$ and B8 has $\approx 7902Hz$.

2.3 Signals

According to [Chakravorty, 2018] a signal may be defined as an observable change in a quality such as quantity. Examples of signals are gestures, images, human voice, sounds, etc. Technically, signals can be represented as a function of time, space or other observation variable that transfers information. In audio, signals are a form of sound representation usually based on electrical quantities such as voltage and expressed in time dimension (Figure 2.9).

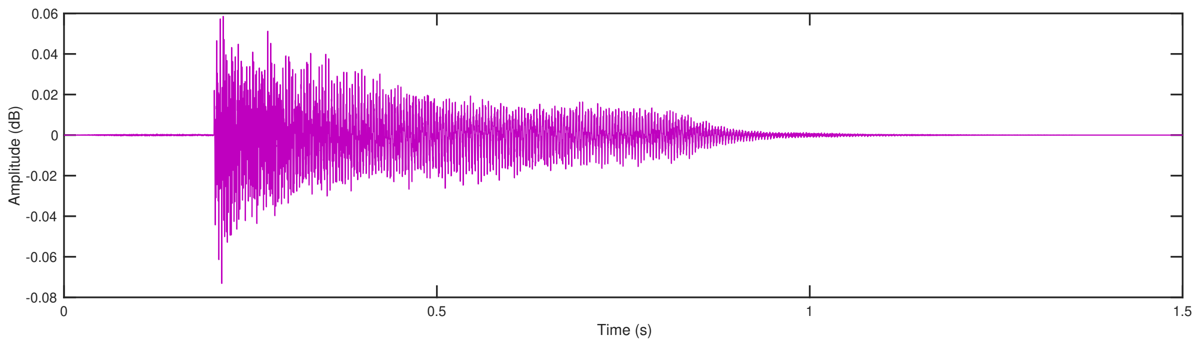


Figure 2.9: Sound signal as a time function: 2 simultaneous piano notes pitch 59 and 79

2.3.1 Fundamental Concepts

There are fundamental concepts about signals in what concerns to signal analysis and processing. Some of them are related with the signal classification depending on their specific characteristics. Some characteristics like periodicity and nature of the independent variable (typically time) are important to understand signal processing concepts. We will present a brief description of the main signal characteristics and classifications possibilities which we think that are more important for a easier understanding of this thesis.

Continuous-Time Signals

A signal is continuous in time if the independent variable (t) is continuous, and theres is a $x(t)$ for all t values in an interval: $\exists x(t) : \forall t,\ t \in \mathbb{R}$. The analog audio signals are defined as continuous time signals(see Figure 2.5-a).

Discrete-Time Signals

The sampling process of an analog signal $x(t)$ leads to a discrete time signal $x[n]$. Instead of a continuous independent variable t, where all the values in an interval $[a; b]$ have a corresponding image $x(t)$, we have only a set of possible values for n and the corresponding images $x[n]$, this way we may consider $n \in \mathbb{Z}$. Considering the same interval $[a; b]$, we have a finite number of possible values for n depending on the the sampling frequency f_s, (see Figure 2.5-b).

Periodic Signals

A signal is considered a periodic signal if it completes a pattern within a measurable time frame, called a period and repeats that pattern over identical subsequent periods. The completion of a full pattern is called a cycle. In time based signals a period is defined as the amount of time required to complete one full cycle. The periodicity is independent of the time nature of the signal, continuous or discrete. For continuous time signals:

$$\tilde{x}(t) = \tilde{x}(t + mT),\ \forall t \in \mathbb{R} \tag{2.7}$$

Equation 2.7 defines a periodic signal where the lowest positive T values is called period of the fundamental frequency, and it is represented by T_0. The fundamental frequency ω_0 is defined, according to the period of the fundamental by:

$$\omega_0 = \frac{2\pi}{T_0}. \tag{2.8}$$

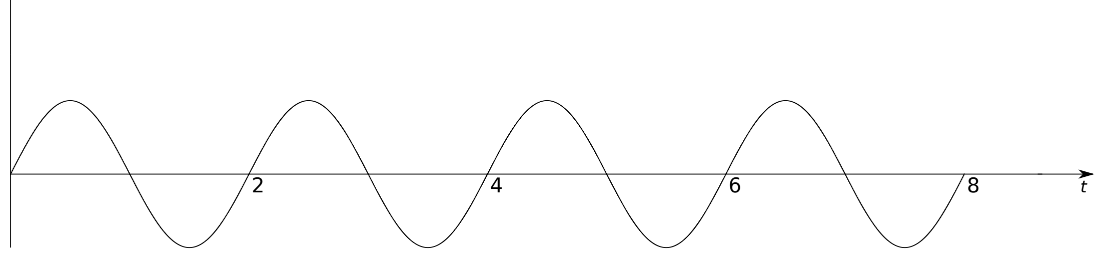


Figure 2.10: Periodic signal with period $T_0 = 2$

In Figure 2.10 is represented a periodic signal $\tilde{x}(t) = \sin(\pi t)$, with fundamental frequency $\omega_0 = \pi$, which corresponds to the period $T_0 = 2$.

The periodicity definition for discrete time signals is similar to the continuous one. A discrete time signal $\tilde{x}[n]$ is said to be periodic with $n, m \in \mathbb{N}$, such that:

$$\tilde{x}[n] = \tilde{x}[n + mN], \; \forall n \in \mathbb{Z} \tag{2.9}$$

The period of the signal is the smallest value of N for which the above relation holds true.

Quasi-Periodic Signals

In mathematics, a quasi-periodic function is a function that has a certain similarity to a periodic function. The discrete signals that have some periodicity properties and follow the representation:

$$\tilde{x}[n] \approx \tilde{x}[n + mN] \tag{2.10}$$

are considered quasi-periodic signals. Quasi-periodic signals are a generalization of periodic signals. The general waveshape of a quasi periodic signal is nearly the same as if it were a periodic signal.

Whereas a sound or music signal has a F0 that is not constant in the time domain, a typical assumption is made: in a very short time interval the signal is assumed stationary. Therefore it is possible to determine the F0 of a non-stationary periodic signal, using the approximation 2.10, in the interval. Figure 2.11 illustrates an example of a quasi-periodic signal, with distinctive periods.

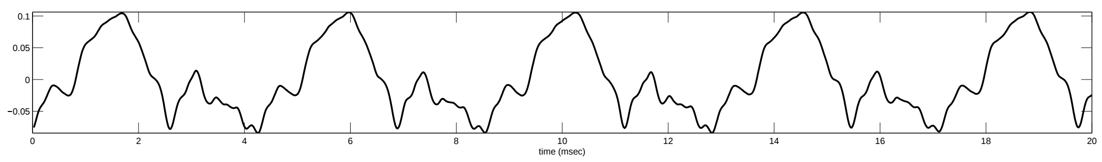


Figure 2.11: Waveform of a quasi-periodic sound signal, of saxophone: F0 = 237Hz T_0 = 4.2ms, adapted from [Reis, 2012]

2.3.2 Signal Processing

Signal processing uses measurements techniques to extract information from a signal that represents some kind of phenomena. It involves the manipulation of real-world signals (for instance, audio signals, video signals, medical or geophysical data signals etc. It can be seen from different perspectives: acoustically, it is a tool to turn measured signals into useful information, to an electrical engineer, it is often restricted to digitization, sampling, filtering, and spectral operations. Signal processing systems include the following.

- Digitization (sampling and quantizing);

- Transforms (Fourier, Laplace, Z);
- Spectral analysis;
- Filtering (time, space frequency);
- Detection, Classification, tracking.

A broader view of modern signal processing can be found by considering its theoretical foundations, which come from mathematics. The signal processing can be applied to continuous and discrete time signals, using similar mathematical foundations. However we will see more detailed the discrete time signal processing that along with quantization transforms a continuous time signal into a digital one. Therefore we will describe the fundamentals and techniques used in Digital Signal Processing (DSP).

2.4 Digital Signal Processing

Digital Signal Processing comprises the application of computational methods and techniques on signals that are digital, i.e. discrete in the independent variable or variables and quantized to analyse, recognize, classify or transform them. It involves the manipulation of real-world signals (audio signals, video signals, medical or geophysical data signals etc.) within a digital computer. To transform natural signals into digital it is used the digitization process that includes analog to discrete conversion and quantization. Nowadays, every representations and most of the ways of communications of sound and image are digital resulting from Digital Signal Processing techniques, such as the JPEG, MPEG, MP3 and GSM.

Audio signal processing is one of the applications of DSP and makes use of several transforms, being the most relevant to our work the Fourier Transform, which will be introduced in the next section.

2.4.1 Fourier Analysis

Independently of the source of a sound wave or even the medium where it propagates itself the particles constituting the medium vibrate back and forth at a given frequency. As mentioned before a sound wave has two major quantities **Period** and **Frequency**, one is the inverse of the other. Thus, a sound wave with a longer period has a lower frequency, whereas a sound wave with a shorter period implies a higher frequency (see Figure 2.12).

Jean-Baptiste Joseph Fourier (1768 - 1830) had the clairvoyance to see that a signal could be represented as a sum or linear combination of harmonically related sinusoidal functions, where each component is called harmonic, with a frequency multiple of the fundamental frequency. According to Joseph Fourier, any periodic signal can be represented by a Fourier series, later demonstrated by P. L. Dirichlet [Oppenheim et al., 1997]. In a simple form, Fourier analysis is the process of decomposing any periodic signal into the

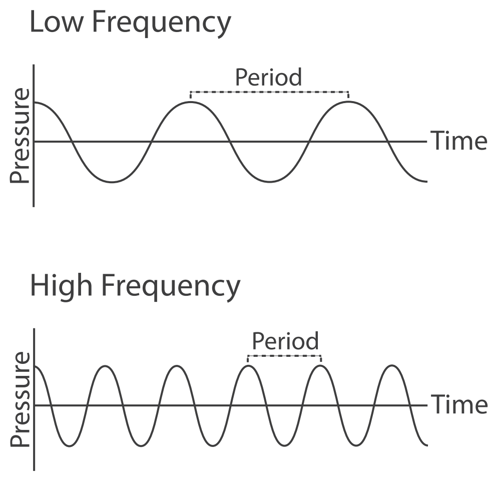


Figure 2.12: Low and high frequencies representation, adapted from [Reis, 2012]

sum of several, even infinite complex exponential functions, that are also a combination of *sines* and *cosines*:

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{2.11}$$

Conversely Fourier synthesis is the process of converting those sinusoidal set of functions into the original periodic function (see Figure 2.13). Consider a simple sinusoidal function sine as a mathematical curve that models an oscillation, the sinusoid is represented as a time function, $x(t)$:

$$x(t) = A\sin(2\pi ft + \phi) = A\sin(wt + \phi), \tag{2.12}$$

where A is the amplitude of the wave, f is the frequency in Hz (oscillations per second), $2\pi f = w$ is the angular frequency, and ϕ is the phase.

As stated before, any periodic signal can be represented by a sum of sinusoidal functions. Considering a square wave, the ideal square wave can be expressed as a sum of only components of odd-integer harmonic frequencies of the form:

$$f_k = 2\pi\,(2k-1) \times f\ ,\ k \in \mathbb{N} \tag{2.13}$$

where f is the square wave and each component with f_k frequency with diminishing amplitude, Figure 2.13 depicts the process.

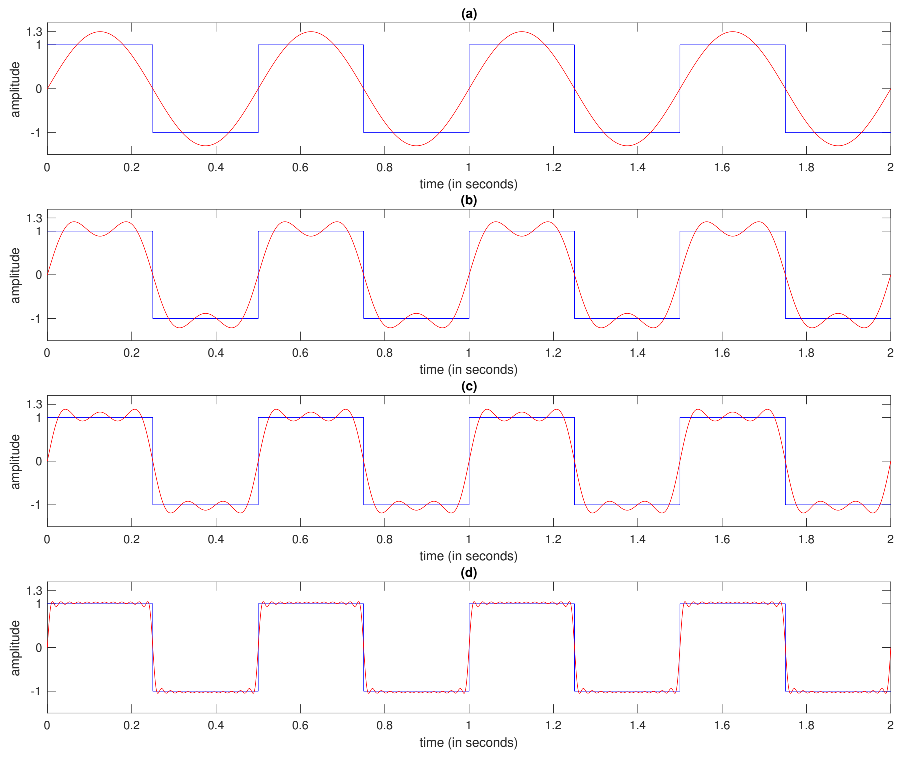


Figure 2.13: Square wave as an approximation of a sum of multiple sinusoids: (**a**) square wave and a sine function with the same F0 (2Hz). (**b**) sum of two sine waves with F0 and an integer multiple of the F0. (**c**) sum of three sine waves oscillating with frequency F0 and integer multiples of the F0. (**d**) a square wave approximation using multiple sine functions.

Figure 2.13-a, shows the square wave in blue and a sin function in red. The red curve is described by the Equation 2.14, both functions have the same frequency F0, $2Hz$.

$$x(t) = 1.3 \sin\left(2\pi 2t\right). \tag{2.14}$$

In Figure 2.13-b, we introduced another sine function with a multiple frequency of the initial sine. The sum of the two sine functions (see Equation 2.15), shows an approximation to the square wave function.

$$x(t) = 1.3 \sin\left(2\pi 2t\right) + 0.42 \sin\left(2\pi 6t\right). \tag{2.15}$$

Depicted in Figure- 2.13-c, is the addition of another partial, the third one (see Equation 2.16), it becomes more clear the progressive approximation to the form of the original square wave.

$$x(t) = 1.3 \sin(2\pi 2t) + 0.42 \sin(2\pi 6t) + 0.24 \sin(2\pi 10t)\,. \tag{2.16}$$

In Figure 2.13-d, we have a sum of almost infinite sines with multiple frequencies and and multiple amplitudes or gains. The resulting sound signal is almost identical to the original, without any difference for human eyes or ear. All those multiple sine wave functions with multiple frequencies are denominated as **partials** or **components**. The major concepts related to this idea applied to sound are: each wave varies with a different frequency; the lowest frequency presented in a certain note is classified as the **Fundamental Frequency** (F0); Fundamental frequency is the inverse of the fundamental period or P0 and corresponds to the perceived pitch. All the waves are called **partials**; frequencies of the partials are mostly limited to integer multiples of the lowest frequency (Fundamental Frequency - F0), forming the **harmonic series**; a **harmonic** is a partial that is an integer number multiple of F0; F0 is also considered an harmonic partial, with a multiplier equals to one. Excluding the fundamental frequency, all the partials are called overtones (over F0).

Nowadays, the Fourier analyses is more than a set of equations or definitions, in tribute to Joseph Fourier. Fourier analysis is considered all the study of signals and systems based on their sinusoidal representation as well as the mathematical techniques used for signal decomposition into a family off sinusoidal functions with a range of frequencies. We have already seen that signals are classified according periodicity an time continuity. Among all the Fourier equations for periodic, non-periodic continuous and discrete functions, one stands up for the context of this thesis: The Discrete Fourier Transform DFT which is used on periodic signals with discrete independent variable, in our case time. The Discrete Fourier Transform is specially suitable for Digital Signal Processing due to discrete and digital computers architecture.

Fourier Series

The Fourier Series (FS) is used for continuous and periodic signals. As it names suggests it is a series, i.e a sum of components. Furthermore any periodic signal represented by a continuous function $\tilde{x}(t)$ can be written as a series, a sum of harmonically related complex exponential functions (can be an infinite series):

$$\tilde{x}(t) = \sum_{k=-\infty}^{\infty} a_k e^{jkw_0 t}, k \in \mathbb{Z} \tag{2.17}$$

where:

$$w_0 = 2\pi F_0 = \frac{2\pi}{T_0}. \tag{2.18}$$

The Fourier Series representation allows the definition of the linear space $L_2([-\pi,\pi])$ of square-integrable functions of $[-\pi,\pi]$ where the set of complex exponential functions form an orthonormal basis. The coefficients of the Fourier Series can be calculated using the analysis equation:

$$a_k = \frac{1}{T_0}\int_{T_0} \tilde{x}(t) e^{-jkw_0 t} dt. \tag{2.19}$$

This way Equation 2.17 is known as synthesis equation and Equation 2.19 as the analysis equation. The synthesis Equation 2.17 can be rewritten in the form::

$$\tilde{x}(t) = a_0 + \sum_{k=1}^{+\infty} \left(a_k e^{jkw_0 t} + a_{-k} e^{-jkw_0 t}\right) \tag{2.20}$$

Taking into account that $a_k^* = a_{-k}$, Equation 2.20 can rearranged as follows:

$$\tilde{x}(t) = a_0 + \sum_{k=1}^{+\infty} \left(a_k e^{jkw_0 t} + a_k^* e^{-jkw_0 t}\right) \tag{2.21}$$

Considering that $a_k \in \mathbb{C}$ and using the Euler Equation 2.11 we can express a_k in its polar form, $a_k = \frac{A_k}{2} e^{j\phi_k}$, by that we obtain the following expression based on a a sum of cosine functions:

$$\tilde{x}(t) = a_0 + \sum_{k=1}^{+\infty} A_k \cos(kw_0 t + \phi_k). \tag{2.22}$$

Equation 2.22 expresses the Fourier Series used for periodic signals. In sound processing any sound signal representable by the previous equations is denominated harmonic sound. Oppositely, the quasi-periodic signals do not have frequencies exactly multiples of the fundamental frequency $F0$, those frequencies are simply referred to as partials, instead of harmonic partials. For this kind of signals and for practical usage, a typical approximation is made, it is considered only a finite number of harmonic partials **H**:

$$\tilde{x}(t) \approx a_0 + \sum_{k=1}^{H} A_k \cos(kw_0 t + \phi_k). \tag{2.23}$$

Fourier Transform

The Fourier transform (FT) is a mathematical transform that decomposes functions depending on space or time into functions depending on spatial or temporal frequency.

Sound waveforms use time as domain, i.e the independent variable of the function that represents the waveform is time. The time-domain representation is useful for several tasks although the time domain information is not the most appropriate for some important tasks that require another kind of information and signal representation. Through the **Fourier Transform** (FT), a signal is transformed into the **frequency-domain**. Considering sound waves, they are represented in time-domain, and a graph shows how the signal value varies along the time variable. Otherwise, by applying the FT we obtain a mathematical function in frequency domain. Thus, plotting a graph we are able to see how the signal lies in the frequency range, or frequency spectrum. Figure 2.14 illustrates the FT domain change and common pairs.

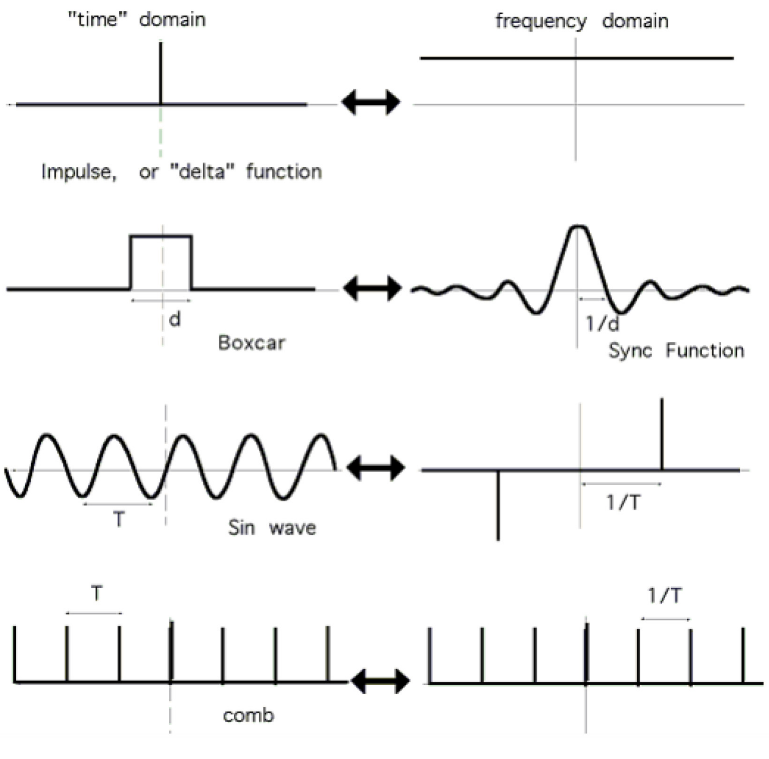


Figure 2.14: Time to frequency domain changes and 4 common pairs, being d a size parameter and T the period.

The term *Fourier transform* refers to both the frequency domain representation and the mathematical operation that associates the frequency domain representation to a function of space or time. Hence, is widely used on tasks that depend of frequency or spectral analysis. The FT is defined as follows for continuous time signals and aperiodic with infinite length:

$$\text{FT}_{\tilde{x}}(f) = \tilde{X}(f) = \int_{-\infty}^{+\infty} \tilde{x}(t)e^{-j2\pi ft}dt. \tag{2.24}$$

As it is possible to conclude by the Equation 2.24 the FT of a function of time is a complex-valued function of frequency, whose magnitude (absolute value) represents the amount of that frequency present in the original function, and whose argument is the phase offset of the basic sinusoid in that frequency.

Discrete Fourier Transform

When we are using discrete time signals the FT is not applicable, instead the discrete Fourier transform DFT) should be applied. This transform can be seen as the sampled version (in frequency-domain) of the Discrete time Fourier Transform output.

As well as FT, DFT is used to calculate the frequency spectrum. However, it uses as input a discrete-time signal. Thus, for a domain transformation of a discrete signal the Discrete Fourier Transform is suitable. Therefore, for Digital Signal Processing the Discrete Fourier Transform is applied. Equation 2.25 shows the mathematical expression used for compute DFT obtaining a frequency domain signal $X[k]$, where k is the independent variable expressed in frequency. Each k is considered a spectral bin:

$$\text{DFT}_{\tilde{x}}[k] = \tilde{X}[k] = \sum_{n=-\infty}^{+\infty} \tilde{x}[n]e^{-j2\pi kn}. \tag{2.25}$$

Due to the obvious fact that computers can only handle a finite number of values it was necessary to rearrange Equation 2.25 applicably to finite time signals. The DFT for finite signals was defined as follows:

$$\text{DFT}_{\tilde{x}}[k] = \tilde{X}[k] = \sum_{n=0}^{N-1} \tilde{x}[n]e^{-j\frac{2\pi}{N}kn}, \quad k = 0, \cdots, N-1 \tag{2.26}$$

where N is length of the signal corresponding to the number of samples used, Figure 2.15, shows the DFT applied to a piano sound signal in time and the correspondent signal resultant in frequency domain.

The Discrete Fourier Transform takes a signal in the so called time domain (where each sample in the signal is associated with a time) and maps it into the frequency domain. The inverse process is also possible using the Inverse Discrete Fourier Transform (IDFT). The inverse Discrete Fourier Transform takes the frequency series of complex values and maps them back into the original time series (discrete time signal). Assuming that the original time series consisted of real values, the result of the IDFT will be complex numbers where the imaginary part is zero and follows the equation:

$$\text{IDFT}_{\tilde{x}}[n] = \tilde{x}[n] = \sum_{k=0}^{N-1} \tilde{X}[k] e^{j\frac{2\pi}{N}kn}, \quad n = 0, \cdots, N-1 \tag{2.27}$$

When a domain swap occurs due to a transformation, it is crucial to calculate the new domain resolution. Typically. each signal is represented using a vector and each vector's element is considered a bin. Each bin as a resolution in time domain and also a resolution in frequency domain. Hence, following the Nyquist-Shannon sampling theorem, [Shannon, 1949] when a DFT of a sound signal occurs a frequency domain signal vector is generated. The range of usable frequencies is majored by the Nyquist frequency $\frac{F_s}{2}$. Therefore N frequency bins is uniformly distributed, the frequency of each spectral bin k is f_k where $f_k = k\frac{F_s}{N}$. Thereby, the frequency resolution of the DFT is $\Delta f = \frac{F_s}{N}$.

Let us consider the Figure 2.15 example, of a piano chord signal. As a sound signal it may be considered a quasi-periodic signal $\tilde{x}[t]$ and it has 4096 samples[1] at a sampling rate is 44100 Hz (similar to digital music like CD quality). After applying the DFT to the signal $\tilde{x}[t]$ in time domain, it is transformed into the signal $\tilde{X}[k]$ in frequency domain. This signal has a frequency resolution $\Delta f = \frac{F_s}{N} = \frac{44100}{4096} = 10.77Hz$. Therefore each two consecutive bins in frequency differ if $10.77Hz$ and the frequency of each spectral bin k is $f_k = k \times 10.77Hz$.

Fast Fourier Transform

To transform a time-domain signal $x[n]$ with N samples using DFT it is required N^2 complex multiplications and $N^2 - N$ complex additions in order to obtain a N sized frequency domain signal $X[k]$. This huge number of operations becomes a very hard and resource consumption computational task, therefore a real time DFT of a considered sized signal is almost impossible. To smooth this problem Cooley and Turkey proposed an algorithm called fast Fourier transform (FFT) [Cooley et al., 1967]. The main idea re-expresses the discrete Fourier transform (DFT) of an arbitrary composite size in terms of smaller DFTs of smaller sizes, recursively, to reduce the computation time to $N \log_2 N$.

The simplest and most common form of FFT algorithm uses radix 2 decimation in time assuming the number of sample N as a power of 2. Each DFT Can be re-written as

[1]The sound signal has 4096 samples, the graph only shows a part of it time and frequency for a better observation.

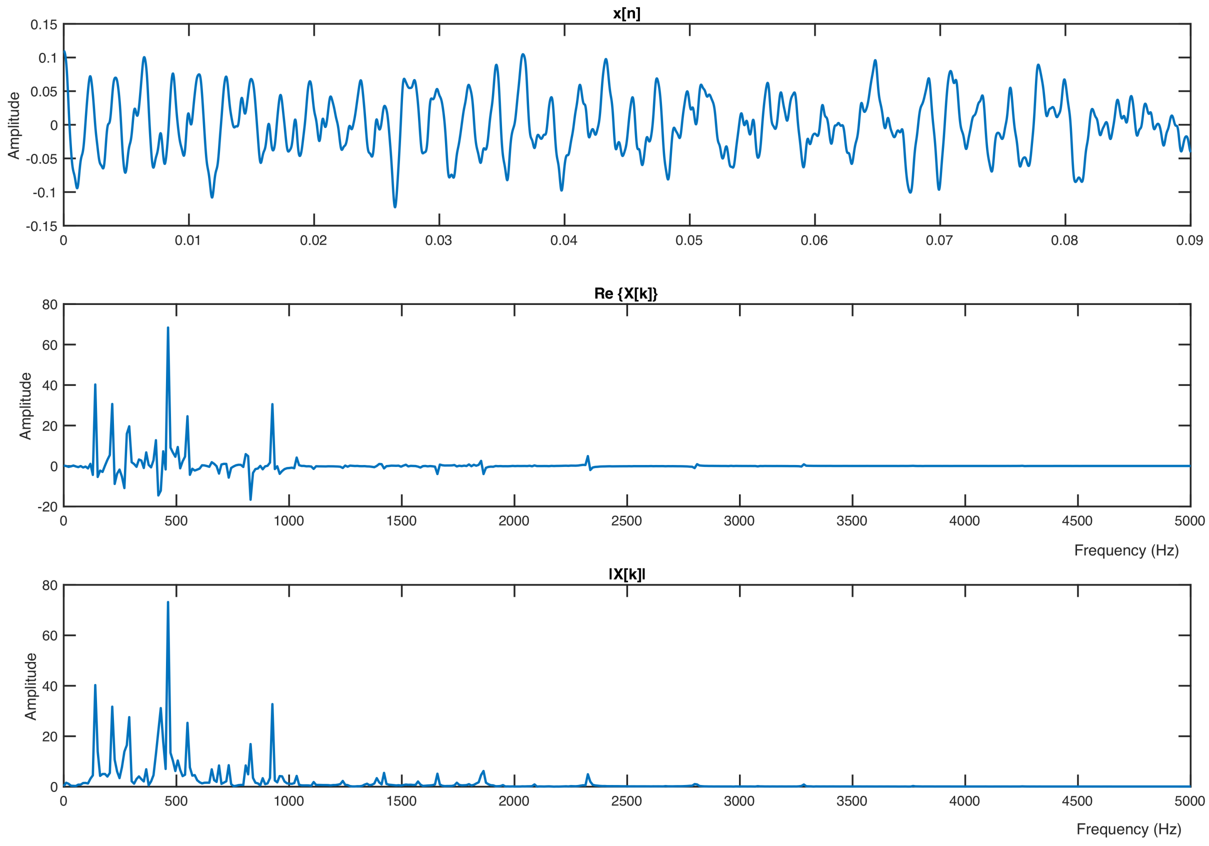


Figure 2.15: Discrete time Fourier Transform - DFT of a piano chord with $90ms$ composed by 3 pitches (38, 49 and 70): (**top**) piano signal in time domain $x[n]$, (**middle**) $\Re\{X[k]\}$ Real part of the DFT of the piano sound and (**bottom**) $|X[k]|$ the module of the DFT. Last 2 graphs in frequency domain expressed in Hz

sum of two summations, equation 2.28 using a variable change and vector decomposition:

$$\begin{gathered} X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}kn} \Leftrightarrow \\ \Leftrightarrow X[k] = \sum_{m=0}^{\frac{N}{2}-1} x[2m]e^{-j\frac{2\pi}{N}k2m} + \sum_{m=0}^{\frac{N}{2}-1} x[2m+1]e^{-j\frac{2\pi}{N}k(2m+1)} \end{gathered} \tag{2.28}$$

the 2 summations can be seen as 2 DFTs of 2 parts of the original time signal:

$$X[k] = \underbrace{\sum_{m=0}^{\frac{N}{2}-1} x[2m]e^{-j\frac{2\pi}{N}k2m}}_{\text{DFT of even indexed part of } x[n]} + \underbrace{\sum_{m=0}^{\frac{N}{2}-1} x[2m+1]e^{-j\frac{2\pi}{N}k(2m+1)}}_{\text{DFT of odd indexed part of } x[n]} \tag{2.29}$$

Thus, using equation 2.29 FFT algoritm first computes the DFTs of the even-indexed inputs and of the odd-indexed inputs, and then combines those two results to produce the DFT of the whole sequence this idea can then be performed recursively until $N = 1$. The schematic algorithm for FFT with radix-2 decimation in time is usually called Butterfly and it is depicted in Figure 2.16

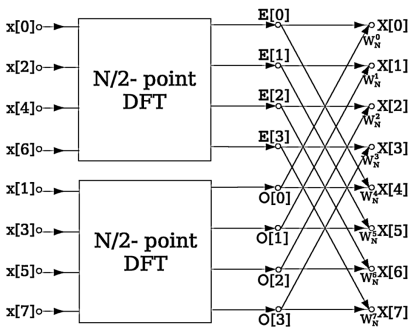


Figure 2.16: Butterfly scheme for implementation of the fast Fourier transform algorithm for $N = 8$

In a nutshell the FFT is an algorithm for fast implementation of the DFT with efficiency. For N structured as a power of 2, the computational effort consumed is proportional to $N \log_2 N$, thus it is a $O(N \log_2 N)$ in what concerns to mathematical operations, allowing DFT computation in real-time.

2.4.2 Convolution

In signal processing the discrete convolution is by far the most commonly used operation, but also most commonly misused and confused. In the maths context, convolution is a mathematical operation on two functions f and g that produces a third function $f * g$ that expresses how the shape of one is modified by the other. The term convolution refers to both the result function and to the process of computing it.

In the DSP context the convolution is defined over discrete time signals, considering two functions or signal $f[n]$ and $g[n]$ defined in time discrete domain where time n is the discrete convolution operation is defined as follows:

$$f[n] * g[n] = \sum_{m=-\infty}^{\infty} f\left[m\right] \cdot g\left[n-m\right] \tag{2.30}$$

we may see the convolution between 2 discrete signals as a process of sliding one mirrored signal, in this case g, and the inner product[2] between that signal for each slide and the other signal f. In DSP the convolution is widely used to perform the output of system to any kind of input, Figure 2.17.

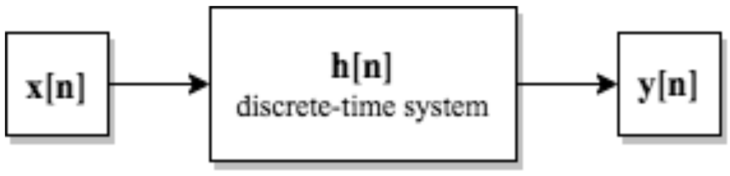


Figure 2.17: Discrete time system with an impulse response $h[n]$

Considering the discrete time system in Figure 2.17, the output $y[n]$ can be calculated through the convolution between the input $x[n]$ and the system impulse response $h[n]$:

$$y[n] = (x * h)[n] = \sum_{m=-\infty}^{\infty} x\left[m\right] \cdot h\left[n-m\right] \tag{2.31}$$

However, in practice this is not a simple way to visualize the output or the influence of the system on the input besides it is also hard to compute. The **convolution theorem** is a fundamental tool to solve some DSP problems. The convolution theorem states that the Fourier transform of a convolution of two signals is the point wise product of their Fourier transforms, this states true for continuous and discrete signals. In a another way, convolution in time domain equals point-wise multiplication in frequency domain, therefore for the system in Figure 2.17 the convolution theorem can be written as follows:

$$Y[k] = F\{x[n] * h[n]\} = X[k].Y[k] \tag{2.32}$$

where[3]:

$$F\{x[n]\} = \mathrm{DFT}_{x[n]}[k] = X[k] \tag{2.33}$$

thus, we may obtain the Figure 2.17 output $y[n]$:

$$y[n] = x[n] * h[n] = F^{-1}\{X[k].Y[k]\}\} \tag{2.34}$$

where F^{-1} is the inverse Fourier transform defined in equation 2.32. This form is often used to efficiently implement numerical convolution by a computer and consequently calculate the resulting output of a system for a particular input.

[2]“.”operator in equation 2.30 means inner product.

[3]“.”operator in equation 2.32 means point-wise product.

2.4.3 Power Spectrum Density

Analysing equations of the Fourier transform (equations 2.25 and 2.24) we conclude that the FT or the DFT have codomain in the set of complex numbers $\mathbb{C}$. A Complex number may be seen as a point in a two dimensional space, known as the *Argand diagram.* Thus, any complex number, $z = a + jb$ is represented as a two coordinates point in the plane space (see Figure 2.18), where a is the real part, and b is the imaginary part often called Cartesian coordinates. j is the square root of -1: $j = \sqrt{-1}$. We can also represent the same complex number using polar coordinates, the magnitude or radial coordinate $|z|$, and the angular coordinate or phase $\phi(z)$. The conversion equations between the two distinct point representations are: $|z| = \sqrt{a^2 + b^2}$ and $\phi(z) = \arctan_2\left(\frac{b}{a}\right)$, where:

$$\phi(z) = \arctan_2\left(\frac{b}{a}\right) = \begin{cases} \arctan\left(\frac{b}{a}\right) & a > 0 \\ \arctan\left(\frac{b}{a}\right) + \pi & b \geq 0, a < 0 \\ \arctan\left(\frac{b}{a}\right) - \pi & b < 0, a < 0 \\ +\frac{\pi}{2} & b > 0, a = 0 \\ -\frac{\pi}{2} & b < 0, a = 0 \\ \text{undefined} & b = 0, a = 0 \end{cases} \tag{2.35}$$

Therefore, any complex number may be represented with Cartesian coordinates or polar coordinates, thus according to Euler´s equation (equation 2.11) any complex exponential can be transformed into a Cartesian representation of a complex number with trigonometrical functions.

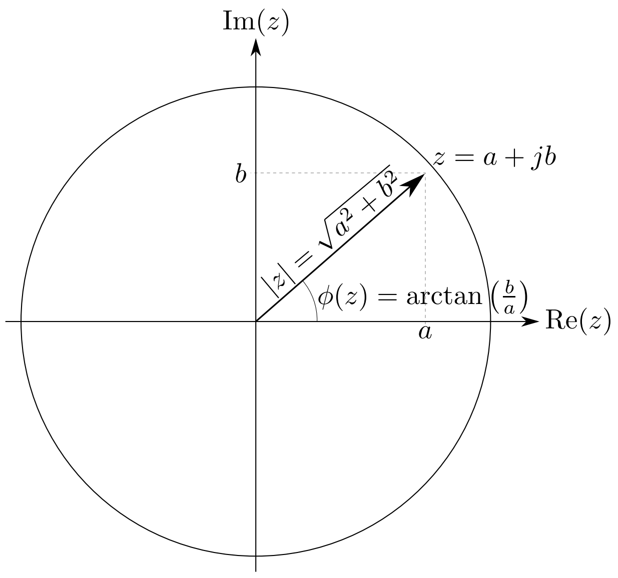


Figure 2.18: Complex plane with Cartesian and polar point representation: magnitude and phase mathematical expression

The power spectrum of a time series describes the distribution of power into frequency components composing that signal [Cooley et al., 1967]. When the signal is finite in time and its total energy is also finite, we are able to compute the energy spectral density or more commonly the **Power Spectral Density** PSD. PSD or simply **spectrum** is a function of frequency that describes how the energy of a signal is distributed along the frequency range. Typically is represented by a two axis diagram or graph, depicting the energy of the signal as a frequency function. Figure 2.19 PSD is computed through squared magnitude of the DFT of a signal $x[n]$ or $|X[k]|^2$ divided by the frequency resolution $\frac{f_s}{N}$.

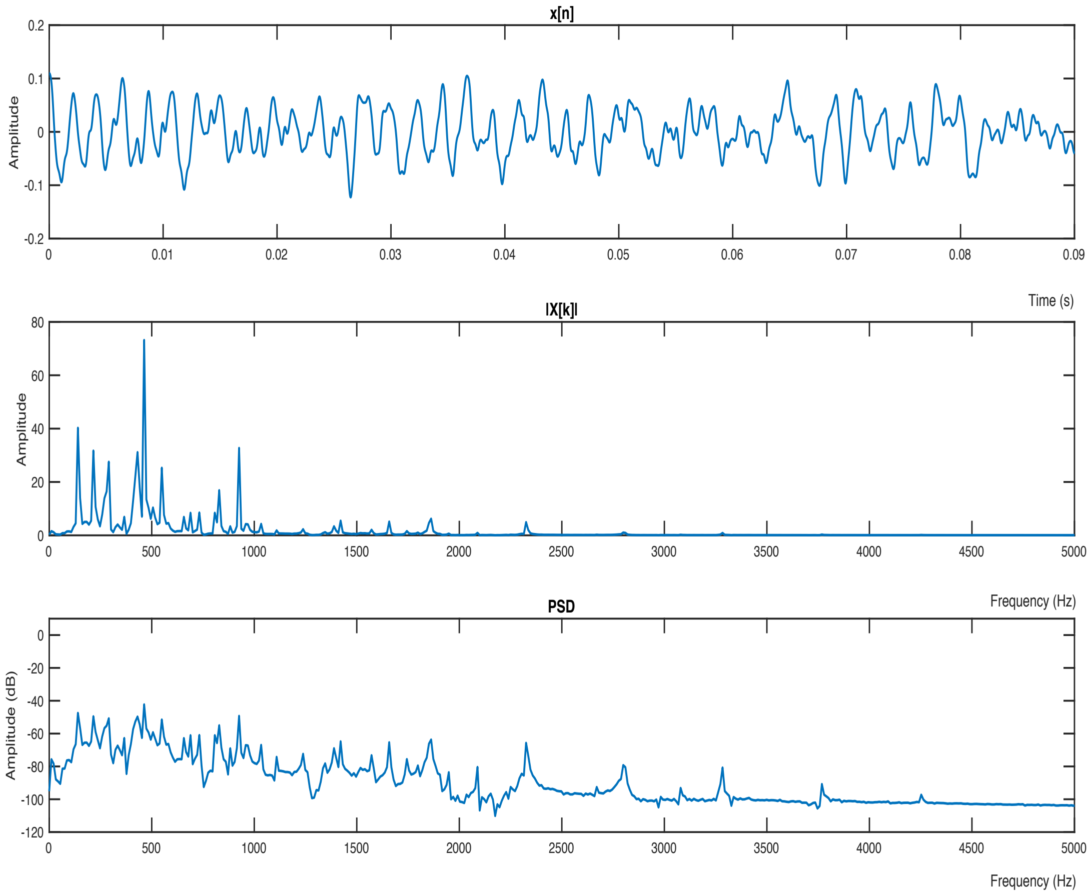


Figure 2.19: A piano chord with $90ms$ composed by 3 pitches (38, 49 and 70): (**top**) piano signal in time domain $x[n]$, (**middle**)$|X[k]|$ the module of the DFT or the magnitude spectrum (**bottom**) PSD - Power Spectrum Density. Last 2 graphs in frequency domain using Hz

In Figure 2.19 (bottom), the PSD is represented using the logarithmic scale for signal amplitude. The “deciBel”(dB) is the logarithmic scale commonly used for representing signal amplitude in the field of signal processing:

$$\mathrm{dB}(|\tilde{X}[k]|) = 20\log_{10}(|\tilde{X}[k]|) = 10\log_{10}(|\tilde{X}[k]|^2). \tag{2.36}$$

A logarithmic scale (or log scale) is a way of displaying numerical data over a very wide range of values in a compact way and it is widely used in signal processing and sound systems, due to its relation to the human earing perception, which is also logarithmic.

2.4.4 Spectral Leakage

The DFT is a data processing tool whose input is a digital signal and whose output is its digital spectral analysis. The discrete nature of the DFT yields intrinsic quantization errors it also assumes that the input signal is periodic and infinite in time. Considering a simple signal like a sine with a frequency of $10Hz$, it is fundamental to calculate the exact number of samples to represent it in order to calculate the exact period. If we apply the DFT using the FFT to a signal that due to some reason is not periodic in discrete time domain, for instance the number of samples used is not a multiple of the period, this leads to a discontinuity and the frequency representation of the signal will be distorted (see Figure 2.20). This effect is known as **spectral leakage** and it describes the process of the leakage of the energy of a frequency bin to the adjacent spreading the signal energy.

The mismatch between the tone of the signal and the chosen frequency resolution (result of sampling frequency and the DFT length) leads to spectral leakage as well as Time-limiting an observation. This effect could interfere with the overall shape of the magnitude spectrum.

2.4.5 Windowing

Windowing is the process of using smoothing windows in time domain to obtain frequency domain signal with spectral characteristics improved, compared to the original signal. In signal processing, the windowing technique is used to minimize the naffest effect in the spectrum caused by the spectral leakage due to the discontinuities of truncated waveforms. The windowing process consists on the point wise multiplication of the time signal by a smoothing window with finite length, defined in samples, and smooth amplitude variation towards zero values at the edges. According to the convolution theorem explained in section 2.4.2, multiplication in one domain is equivalent to convolution in the other domain (i.e. multiplication in time domain is equivalent to convolution in frequency domain). Thus, the resulting signal Y_k in frequency domain of a time windowed signal after applying the DFT may be computed using the Equation 2.37, where a_n represents the used window.

$$Y_k = \sum_{n=0}^{N-1} a_n \tilde{x}_n W_N^{nk}, \quad W = e^{-j2\pi}. \tag{2.37}$$

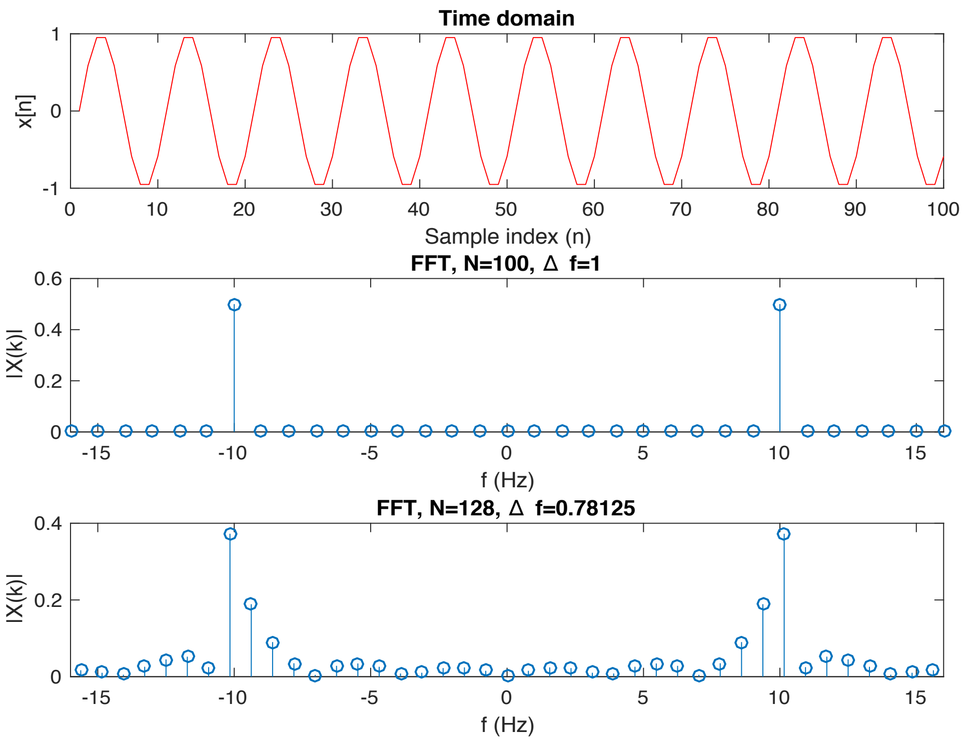


Figure 2.20: Spectral leakage illustration for a sine-wave due to windowing and sampling: (**top**) Sine-wave of $10Hz$, sampled at a $100Hz$ sampling rate, (**middle**) magnitude spectrum of the signal using FFT with 100 samples and (**bottom**) the magnitude spectrum of the same original signal using the FFT with more samples, however not a multiple of the period leading to a spectral leakage

In signal processing the most commonly used windows are triangular rectangular Hanning, Hamming, Nuttal and Blackman. Typically, window functions are mathematical functions that are zero-valued outside of some chosen interval, normally symmetric around the middle of the interval, usually near a maximum in the middle, and usually tapering away from the middle. An important study about the usage of different window functions was presented by [Harris, 1978].

Windowing, changes the shape of the signal in the time domain, as well as affects the spectrum, in Figure 2.21 is depicted the windowing process using the Hanning window.

2.4.6 Signal's Properties Relation

There are some parameters used in signal processing to change the signal domain from time to frequency using DFT that have significant effect on the signal spectral analysis. Among them are the window sampling frequency, the length of the window in samples (N) and the DFT or FFT number of points (typically a power of 2). These parameters have a direct impact in time resolution and frequency resolution for signal analysis.

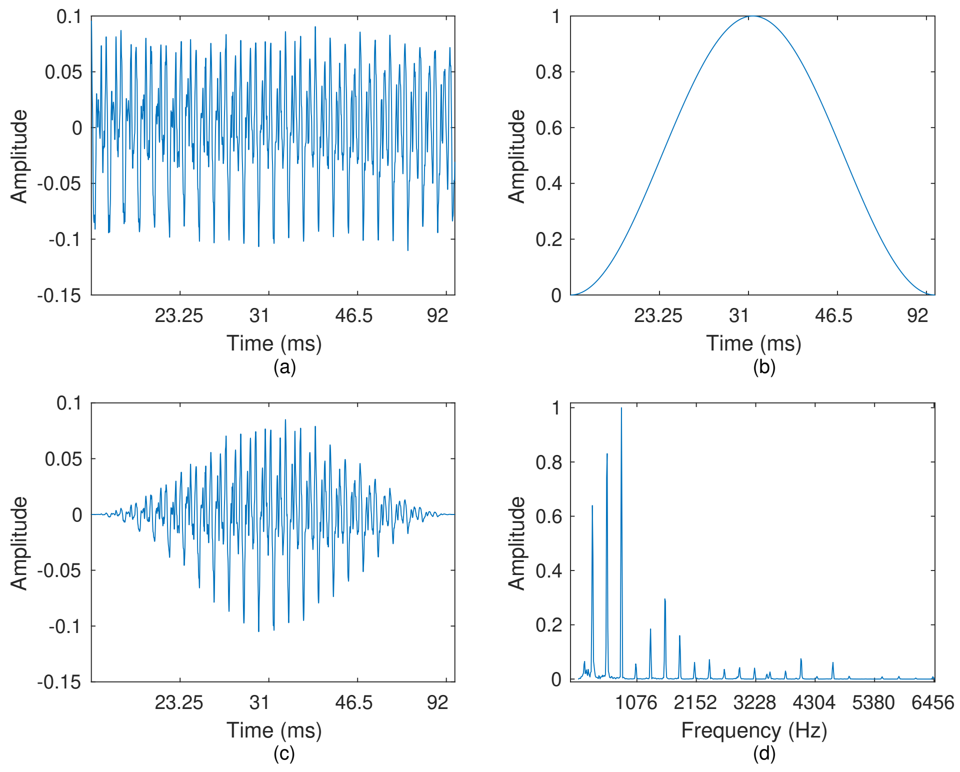


Figure 2.21: Windowing a sound signal: (**a**) input time signal piano note, (**b**) Hanning window, (**c**) resulting windowed signal, (**d**) Resulting signal after DFT

The length of a discrete time signal (number of samples n) depends on the sampling rate F_s and portion of time of the signal that we consider for analysis t:

$$n = F_s \times t \tag{2.38}$$

as explained before, the DFT (Discrete Fourier Transform) transforms the time signal into a signal in the discrete frequency domain where k corresponds to a bin with a frequency. Thus, the signal **frequency resolution** is inversely proportional to the number of samples N as follows:

$$\Delta F = \frac{F_s}{N}, \tag{2.39}$$

being N the DFT size. **Time resolution** follows the equation:

$$\Delta t = \frac{N}{F_s}. \tag{2.40}$$

Consider a typical example for audio processing: a piano music with a duration of 1 seconds, acquired using a sampling rate of 44100 samples per second, and we apply a DFT window size is 4096 samples.

- Time $= 1 \times 44100 = 44100$ samples.
- $\Delta F = \frac{44100}{4096} = 10.7$, each bin will correspond to 10.7 Hz.
- $\Delta t = \frac{4096}{44100} = 0.09$, the time window of the stationary is 0.09 seconds long.

With this approach and parameters we have a frequency precision of $10.7Hz = \Delta F-$, this means that we are only able to distinguish frequencies that differ more than ΔF, frequency resolution. In terms of piano pitches it causes constrains for pitches lower than 54, however we are able to analyse portion of sound with 0.09 seconds. To have a greater frequency resolution we have to increase the size of the DFT and obviously the amount of time analyzed, which may cause difficulties to recognize sounds and pitches with short-time duration. Inversely, we can decrease the size of the DFT and the amount of time analyzed in one frame, however we will loose precision in frequency domain it will be more difficult to distinguish near frequencies. This dilemma demands a compromise solution between time and frequency resolution.

2.5 Pitch and Fundamental Frequency

The frequency of a sound stimulus refers to the number of cycles in the sound wave that occur in 1 second, it is an objective concept that is possible to measure accurately. The perceptual correlate of frequency is pitch and it is an auditory sensation, tones with lower frequencies are heard as being low in pitch, whereas tones that have high frequencies are heard as high in pitch. We define frequency as the number of cycles in a sound stimulus that occur during 1 second. Pitch is the subjective experience of sound that is most closely associated with the frequency of a sound stimulus. Despite of the sound wave vibrations may be measured to obtain a precise frequency, pitch is not a feature of sound waves. The human brain is fundamental for mapping quality pitch perception. The brain perceives a pitch by the fundamental frequency or the periodicity of an audio signal however pitch of complex tones may become ambiguous: Consider a signal with two pure tones at $1000Hz$ and $1300Hz$, we might perceive a *"missing fundamental"* by hearing those two frequencies and an additional one, created by the difference of the two signals. This phenomenal also called phantom fundamental is illustrated in Figure 2.22 where besides the 2 original pitches, appears an additional one corresponding to $300Hz$.

According to the American National Standard Institute (ANSI), pitch is defined as "that auditory attribute of sound according to which sounds can be ordered on a scale extending from low to high". Although it still is a subjective definition, [Hartmann, 1997] gives a more objective and technical definition: "sound has certain pitch if it can be reliably matched by adjusting the frequency of a sine wave of arbitrary amplitude".

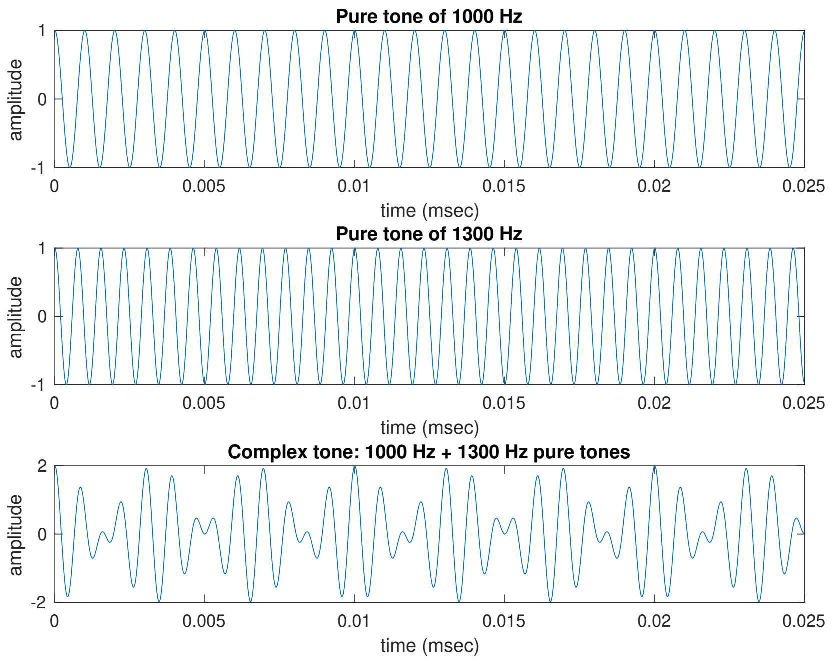


Figure 2.22: Missing fundamental phenomenal at 300 Hz, adapted from [Reis, 2012]

For the majority of the authors, the term **F0 estimation** is equivalent to **pitch estimation**. However, some consider F0 estimation, the process of extraction the exact frequency components of a given signal and the second one the process of estimation the the musical tones or pitches that are presented in a sound signal. Along this dissertation we will tackle the problem of multi-pitch estimation and some times we will refer to it as **Automatic Music Transcription**, AMT.

2.6 Single Pitch Estimation

Sound signals may be *polyphonic* if there are more than one sound playing simultaneously or *monophonic* when there is only one note at the same time [Klapuri and Davy, 2006]. For instance one piano chord is a polyphonic sound and one piano key pressed is considered a monophonic one. Therefore, Single Pitch Estimation is applied to sound signals typically short-time signals where is presented only one tone or note. An important approximation was stated by [Yeh, 2008a], the observed signal can be expressed as a sum of a quasi-periodic part $\tilde{x}[n]$ and the residual $z[n]$:

$$x[n] = \tilde{x}[n] + z[n] \approx \sum_{k=1}^{H} A_k \cos(kw_0 t + \phi_k) + z[n] \tag{2.41}$$

where Equation 2.23 is used for approximation.

The single pitch estimation problem or the single F0-estimation problem consists on estimating the the pitch or fundamental frequency of a sound signal in a small time part or frame of a sound signal. The focus will be on the quasi-periodic part $\tilde{x}[n]$ with high periodicity and harmonicity and the residual part $z[n]$ will be neglected. The major difficulties and also the most common errors are directly related to the sub-harmonic and super-harmonic. In both cases the frequencies are strongly related with the correct fundamental frequency (F0) being sub-multiples or multiples.

One common categorization for single pitch estimation methods is based on the signal analysis domain: some methods use time domain and others lie on the frequency domain. The time domain methods depend on estimate or find the correct period of the signal to estimate the frequency, whereas the frequency domain methods are based on a spectral analysis. The majority of the algorithms ignore the source model expressed in Equation 2.41, they often try to extract the periodicity or frequency directly in time domain or from spectral analysis (frequency domain).

2.6.1 Spectral-location Approaches

Time-domain methods take advantage of the periodic properties of signals to perform fundamental frequency estimation. This methods are based on pattern matching along time domain, searching for a repetitive pattern in $x[n]$. Considering the original signal $x[n]$ and itself delayed in time $x[n + \tau]$, the main goal is to find out the lowest τ that produces the highest similarity or the best correlation between the 2 signal versions. This process is usually carried out using point wise multiplications to perform correlation and the highest correlation value found corresponding to the lower delay is considered the signal period.

Autocorrelation

The correlation function is statistical function that gives the statistical correlation between random variables, for the same purpose is used the cross-correlation in signal processing, and it measures the similarity level between to signals, this way **autocorrelation** function (ACF) measures similarities in the signal, it is a cross-correlation between the signal and a delayed version of itself. Mathematically, ACF is defined as function of τ and it is computed as the sum of the product between a signal $x[n]$ of finite duration L and its delayed version $x[n+\tau]$, as a function of the lag τ:

$$\mathrm{ACF}[\tau] = \frac{1}{L-\tau} \sum_{n=0}^{L-\tau-1} x[n]x[n+\tau]. \tag{2.42}$$

As expected, sound signals have high values of ACF for τ multiples of the fundamental period (including fundamental period) due to the quasi-periodic properties of sound signals. After computing the autocorrelation function the τ corresponding to the highest peak, excluding zero, is chosen for the estimated value of the fundamental period and corresponding fundamental frequency. Although this method denotes high sensitive to several aspects as resonance and others, thus many post-processing techniques are added [Hess, 1983] to the autocorrelation.

Average Magnitude Difference

The Average Magnitude Difference (AMDF) was used by [Ross et al., 1974] as a pitch extractor. The main idea is to compute the average of the distance between $x[n]$ and $x[n+\tau]$, the signal and itself with a variable time lag τ and this way find the similarities or in this case dissimilarities between the 2 patterns.

$$\mathrm{AMDF}[\tau] = \frac{1}{L-\tau} \sum_{n=0}^{L-\tau-1} |x[n]-x[n+\tau]|. \tag{2.43}$$

For sound signals the AMDF minima correspond to high similarities between the signal and its delayed version, those minima occur for τ values equal to fundamental period or multiples, the deepest local minimum is selected as the estimated fundamental period.

Squared Distance

The Squared distance Function (SDF) is similar to AMDF it is also a function of τ:

$$\mathrm{SDF}[\tau] = \sum_{n=0}^{L-\tau-1} (x[n]-x[n+\tau])^2. \tag{2.44}$$

The average value of the SDF is also used for the same purpose:

$$\mathrm{ASDF}[\tau] = \frac{1}{L-\tau} \sum_{n=0}^{L-\tau-1} (x[n]-x[n+\tau])^2. \tag{2.45}$$

Both functions show dips for higher similarities between the original signal and the delayed version.

This analytic method was used by [De Cheveigné and Kawahara, 2002] on the well known YIN algorithm for speech and music fundamental frequency estimation. The SDF was used along with some adaptations as a threshold for average normalization over shorter values of τ, helping reducing minima at τ near zero and, thus, avoiding super-harmonic errors. This method is denominated the **cumulative mean normalized difference function**. Both referred methods AMDF and SDF are strongly related to the autocorrelation function. [Hess, 1983] explained and demonstrated that both methods are error prone when submitted to intensity variations, noise and low-frequency spurious signals.

Cepstrum

The first paper on **cepstrum** [Bogert, 1963] defined it as "the power spectrum of the logarithm of the power spectrum". The original application was the detection of echoes in seismic signals, where it was shown to be greatly superior to the autocorrelation function, because it was insensitive to the colour of the signal.

Actually the Cepstrum is computed taking the Inverse Fourier Transform (IDFT) of the logarithm of the spectrum of the signal $x[n]$, nowadays known as real Cepstrum:

$$c(\tau) = \mathrm{IDFT}\{\log\left(|\mathrm{DFT}\left(x[n]\right)|\right)\}. \tag{2.46}$$

However, "complex cepstrum" was also defined by [Oppenheim and Schafer, 2004] as a sequence $x[n]$ is calculated by finding the complex natural logarithm of the Fourier Transform of $x[n]$, then the inverse Fourier Transform of the resulting sequence. It takes in account the phase of the signal and it can be shown that the real cepstrum is the even part of the complex cepstrum. One of the earliest applications of cepstrum was determining the voice pitch of voiced speech [Noll, 1967]. It is used in signal processing as a tool to estimate fundamental frequency. "When a signal exhibits a periodicity which is denoted by sinusoidal peaks in its spectrum, it makes sense to apply again the Fourier analysis on the observed spectrum for analyzing the underlying periodicity." There are several cepstrum varieties: a complex cepstrum, a real cepstrum, a power cepstrum, and phase cepstrum. The analysis and operations based on cepstrum are denominated **cepstral analysis**, the name "cepstrum" was derived by reversing the first four letters of "spectrum" as well as another terms derived from spectrum: **cepstra**, **quefrency** or **liftering** [Oppenheim and Schafer, 2004].

When we apply the cepstrum to a time signal, it is transformed into another domain and it becomes function of another independent variable called **quefrequency**. The quefrenquency is a measure of time however, the resulting signal is not in time domain. If we consider a time sound signal acquired with a sampling rate of $44100Hz$ and applying cepstrum we find a significant peak in 100 samples, it means that our signal has a fundamental frequency component $\frac{44100}{100} = 441Hz$. The peak occurs in the cepstrum due to their periodicity in the spectrum. Thus that period corresponds to the fundamental frequency, since harmonics are integer multiples of the fundamental frequency.

The methods used for single-pitch estimation based on cepstrum and on ACF use an harmonic location of the magnitude spectrum for the frequency partials [Klapuri, 2004a]

Both ACF based and cepstrum-based single-F0 estimation methods are implicit realizations of a model which emphasizes frequency partials at harmonic locations of the magnitude spectrum [Klapuri, 2004a]. If we rewrite ACF equation 2.42 using Fourier spectrum $X[k]$ of a real-valued signal:

$$r(\tau) = \frac{1}{K}\sum_{k=0}^{K-1}\left[\cos\left(\frac{2\pi\tau k}{K}\right)|X[k]|^2\right] \tag{2.47}$$

where K represents the length of the analyzed frame, we obtain:

$$r(\tau) = \text{IDFT}\{|\text{DFT}\,[x[n]]\,|^2\}, \tag{2.48}$$

where we are able to see the similarities between $r(\tau)$ (Equation 2.48) and $c(\tau)$ expressed in equation 2.46. The cepstrum definition $c(\tau)$ is analog to $r(\tau)$ substituting the power of 2 by a logarithm.

In Figure 2.23 is shown the four time-domain methods based on the four functions presented on this section: Autocorrelation, AMDF, SDF and cepstrum. All the 4 methods for single pitch estimation were applied to a piano single pitch sound with a frequency of $F0 = 440Hz$ and a corresponding period $T = 2.3ms$ represented in graphs with a vertical dashed line. We are able to sse the efficiency of the 4 methods for the correct estimation of the fundamental period for this particular case. Despite the existence of a wide range of methods based on the similarity between a short-time signal and its delayed version (ACF, SDF, AMDF, etc.), there still remain some ambiguities and problems using them. The main difficulty is to choose the best period to be estimated one. This ambiguity is a problem because all the multiples of the a period remain a period and are possible choices for selection.

2.6.2 Spectral-interval Approaches

Periodic time signals, (not only sinusoidal) have periodicity in frequency domain. This property can be observed in the magnitude spectrum. The period in the frequency domain is the signal fundamental frequency F0. The spectral domain approaches estimate F0 or pitch analysing the spectrum. There are 2 typical methods in this type of approach: one is measuring the distance in the frequency axis of the spectrum between relevant peaks, the other is based on the formulation of the salience of the F0 as a function of hypothetical partials. We can see the F0 as the greatest common divisor of the frequencies of all the harmonics. The methods based on Frequency domain for F0 estimation analyse or operate on the spectral representation of the original sound signal transformed using the Fourier Transform.

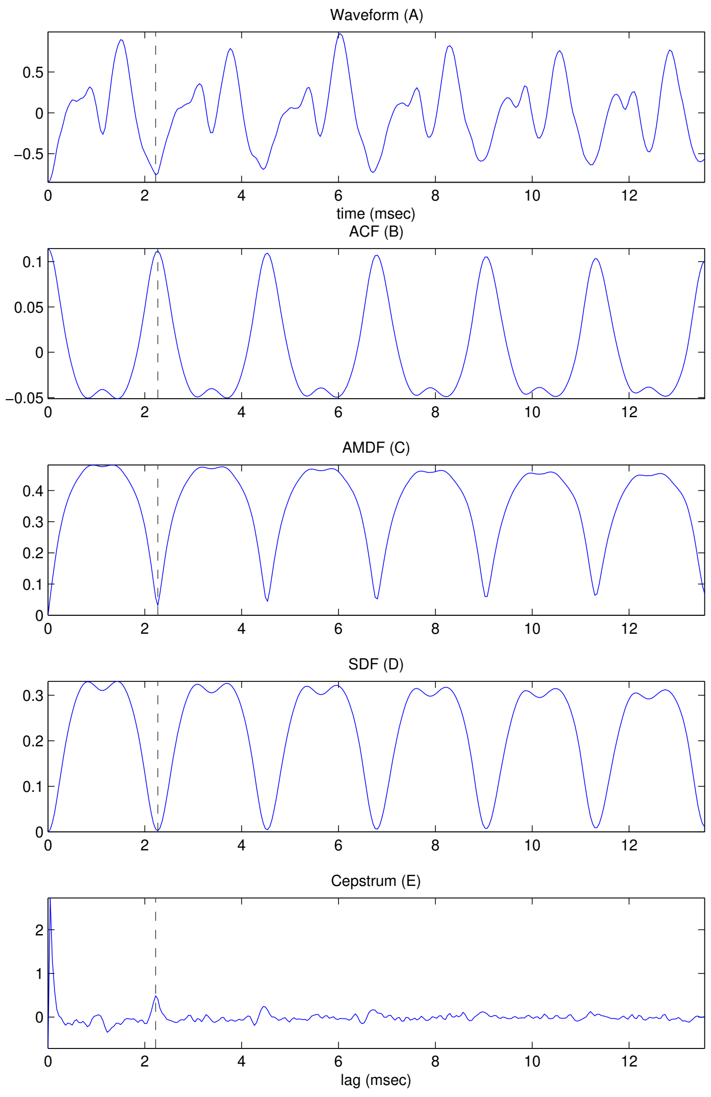


Figure 2.23: (**A**) time signal; (**B**) autocorrelation function; (**C**) average magnitude difference function; (**D**) squared difference function; and (**E**) cepstrum. Figure adapted from [Reis, 2012], page 26.

Autocorrelation in spectrum

Spectral autocorrelation is analog to the time autocorrelation function (ACF) defined in Equation 2.42. The main idea relies on searching for repetitive patterns in a signal. However the analyzed domain is the frequency domain. Thus, the search will take place in the function spectrum. Therefore, the search for pattern match will be made in the frequency domain between the signal transformed $X[k]$ and not its delayed version because we are not in the time domain, but its shifted version $X[k+m]$. The ACFS or the ACF in the spectrum domain can be defined as follows:

$$\mathrm{ACFS}(m) = \frac{2}{K-2m} \sum_{k=0}^{\frac{K}{2}-m-1} |X[k]| \times |X[k+m]|. \qquad (2.49)$$

where K is the length of the magnitude spectrum and, by taking in account that only a part of the spectrum is used, the part with negative frequencies is excluded. The local maxima of the function $\mathrm{ACF}(m)$ for $m > 0$ correspond to high values of autocorrelation. Typically a maximum occurs when shift equals to $m = F0\left(\frac{K}{F_S}\right)$.

Spectral compression

The main idea is based on an frequency-warped spectra built using compression in frequency. An histogram is made counting the contribution of each peak of the spectrum to the related F0s that are also common divisors of its frequency. This is called the Schroeder histogram [Schroeder, 1968], that uses the count process of the contributions of each spectral peak to estimate the F0 of the higher harmonics. This process uses weights for the spectral components according to their magnitudes: "harmonic product spectrum uses the log power spectrum and harmonic sum spectrum uses the linear spectrum. By summing compressed spectra, the energy of higher partials becomes concentrated on distinct peaks. The absolute maximum infers the related F0", [Reis, 2012].

Harmonic matching

The Harmonic matching approaches uses pattern matching techniques to estimate the F0. The principal idea is based on the comparison of the observed spectrum with a given spectral harmonic pattern based on a F0 hypothesis. The spectral pattern used for the comparison process can be a specific model or an harmonic comb without specifying the magnitudes used. The pattern matching process measures the similarity. The similarity between the spectral of the analyzed signal and the fundamental frequency considered as an hypothesis is measured using the correlation between the harmonic combinations and the analysed spectrum [Martin, 1982, Brown, 1992], or calculating the distance between the frequencies of the harmonics and the frequencies of the matched peaks finding the minimum. [Goldstein, 1973, Duifhuis et al., 1982].

2.6.3 Unitary model of pitch perception

Perceptual hearing models are used to simulate the human hearing system and the way it identifies a sound pitch or its fundamental frequency. There are some methods based on the human hearing system to build a usable model. An unitary model was introduced by [Meddis and Hewitt, 1991a, Meddis and Hewitt, 1991b]. This model consists on sequential processing stages such as frequency selectivity, half-wave rectifications and filtering processes and it measures the period of a signal envelope, in time domain to estimate the F0 of the analysed signal. The unitary model is accepted and used as a psychoacoustically

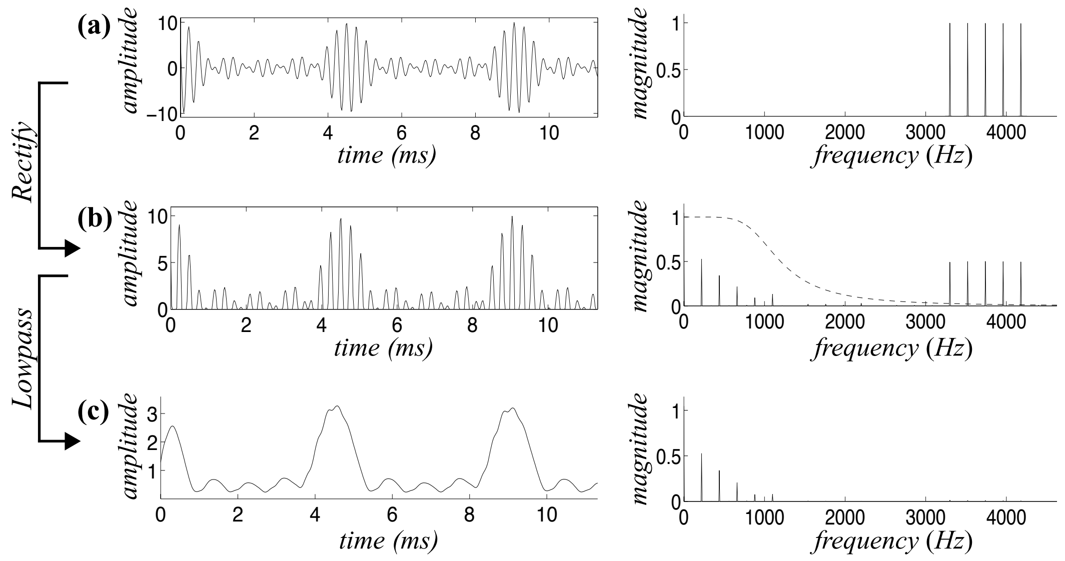


Figure 2.24: From top to bottom: (**a**) waveform of a signal containing the harmonics from 15 to 19 of a sound with F0 = 220Hz; (**b**) signal half-wave rectification; and (**c**) resulting signal from lowpass filtering. Dashed line in (b) represents the frequency response of the lowpass filter. (Adapted from [Klapuri, 2004a], page 27).

valid mid-level representation [Klapuri and Astola, 2002]. It is described as set of stages: it starts with a bank of band-pass filters, then follows a half wave rectification (HWR) and low pass filter process that simulates the convention of cochlear movements into neural impulses. Then, a periodicity estimation by frequency channel using ACF and, finally, a summary aoutocorrelation function (SACF) is used. In Figure 2.24 ([Klapuri, 2004a], page 27) is illustrated the second step that comprises the half wave rectification and the low pass filtering. The half-wave rectifier function is represented as follows:

$$\mathrm{HWR}\,(x) = \frac{x + |x|}{2} \quad = \quad \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} . \tag{2.50}$$

The final step of the unitary model uses the **summary autocorrelation function** (SACF) whitch is computed by summing the ACF across channels. The highest value calculated by the SACF is the estimated **perceived pitch**.

2.7 Multi-Pitch Estimation

Multi-pitch Estimation algorithms are used to analyze and estimate the pitches presented in a sound signal, typically a short-time signal, that includes more than one pitch or more then one harmonic source in the same instant. According to [Yeh et al., 2010] any multi-pitch signal can be written as a sum of harmonic sources $Y_m[n]$ and a residual $z[n]$, where M is the number of harmonic sources:

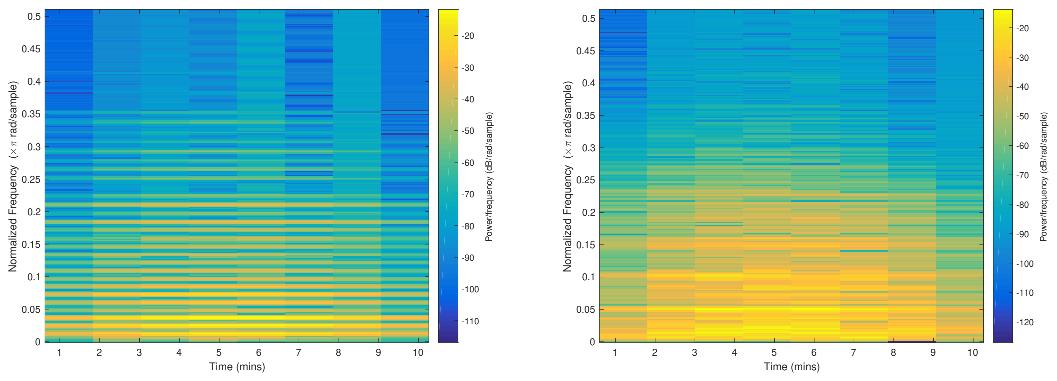
$$y[n] = \sum_{m=1}^{M} Y_m[n] + z[n], M > 0. \tag{2.51}$$

where n is the discrete time index, M is the number of harmonic sources and $z[n]$ is the residual. The residual $z[n]$ can be explained by background noise, spurious components or inharmonic partials. Applying Fourier series to the multi harmonic sources signal we obtain Equation 2.52.

$$\begin{aligned} y[n] &= \sum_{m=1}^{M} \left\{ \sum_{k=1}^{\infty} A_{m,k} \cos(k\omega_m n + \phi_{m,k}) \right\} + z[n] \\ &\approx \sum_{m=1}^{M} \sum_{k=1}^{H_m} A_{m,k} \cos(k\omega_m n + \phi_{m,k}) + z[n]. \end{aligned} \tag{2.52}$$

In the context of practical use, a common approximation is made: a quasi periodic signal is approximated to a finite number of sinusoids H.

Figure 2.25: Comparison between two spectrograms of monophonic and polyphonic signals.



(a) Spectrogram of a monophonic piano signal

(b) Spectrogram of a polyphonic piano signal.

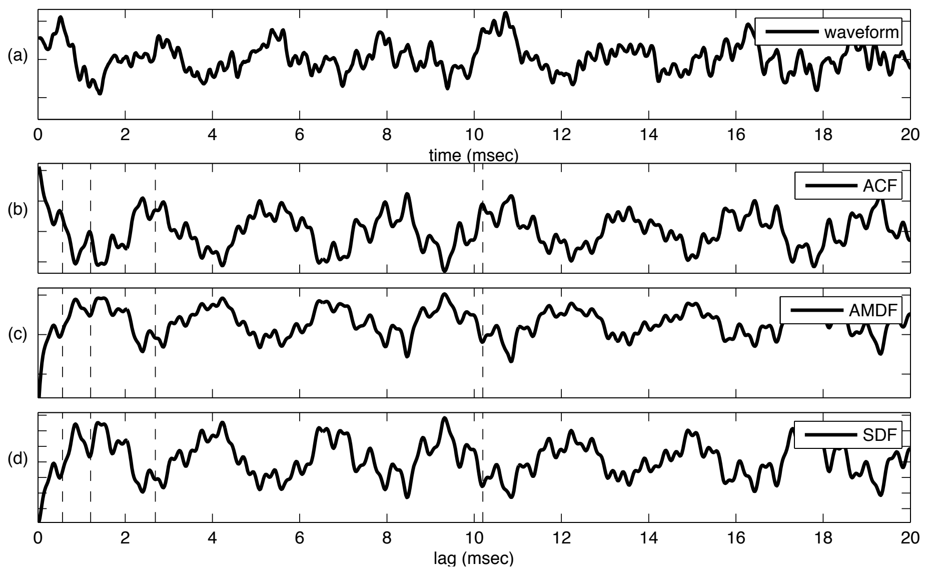


Figure 2.26: Three time-domain salience functions for a polyphonic signal containing four harmonic sources. The correct periods are marked by vertical dashed lines (from [Yeh, 2008a], page 19).

2.7.1 Complexity

The problem complexity of polyphonic music signals is far superior to monophonic music signals. Figure 2.25a shows the representation of a spectrogram of a monophonic piano signal and Figure 2.25b shows te representation of a spectrogram of a polyphonic piano signal with 4 pitches or 4 harmonic sources. Comparing the 2 spectrograms, we are able to see that the polyphonic signal has much more frequencies than the monophonic: the vertical axis has more color changes.

Extracting multiple pitches from music signals is a complex task due to some known factors as: overlaping partials, beating, transients and reverberation. The different musical instruments produce sounds with diverse characteristics, and this is reflected on the spectral diversity. Moreover, there is the identification related with the number of harmonic sources presented in the observed sound. The noise model is also more complex than in monophonic signals and, in real music signals, can be found unpitched sounds like drums or other percussion sounds. The majority of the single-pitch estimation methods struggle when applied to polyphonic sounds. A study conducted by [Yeh, 2008a], page 19 shows the poor performance of several single-F0 estimation methods analysing polyphonic sounds. Figure-2.26 illustrate some time-time domain methods and Figure 2.27 shows the performace of frequency-domain methods.

Overlapping partials

In polyphonic sound signals interference and overlapping between partials may occur either in time domain, or either in frequency domain. This process occurs when two

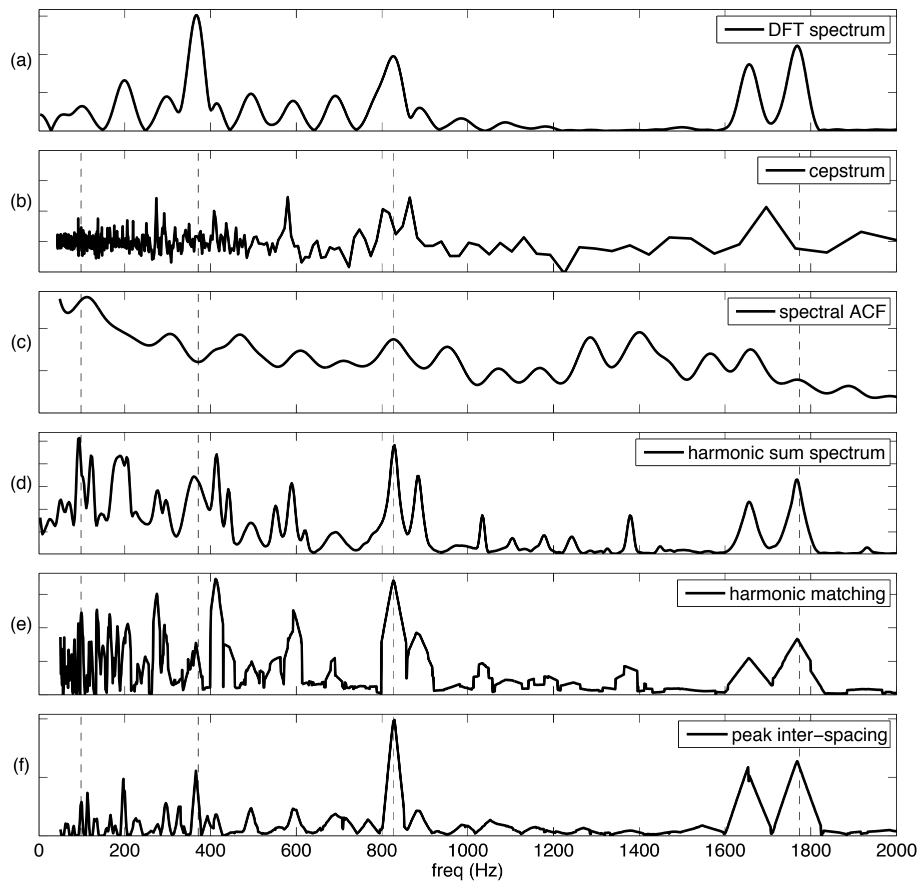


Figure 2.27: Five frequency-domain salience functions applied to a polyphonic signal with 4 harmonic sources. Vertical dashed lines mark the fundamental frequencies, adapted from [Yeh, 2008a]

different sources with fundamental frequencies F_1 and F_2 are harmonically related as in Equation 2.53:

$$F_1 = \frac{m}{n} F_1, \quad n, m \in \mathbb{N}. \tag{2.53}$$

When there is an m and an n that makes the preposition true, every n^{th} partial of the source 1 overlaps every m^{th} partial of source 2 [Klapuri, 1998]. This frequently happens when the sources are harmonically related to each other, since it could result in partial collisions. When dealing with multi-pitch signals most of the musical notes are harmonically related, which frequently results in overlapping partials. Another issue is that when fundamental frequencies of two notes are multiples of each other, the partials of the higher note can overlap completely with those that are from lower note. [Parsons, 1976] addressed the problem of separating the voice of a vocalist speech, by trying to detect the overlapping components, based on three tests: spectral peak symmetry, distance and well-behaved phase. This technique is restricted to two voices and relied on the sinusoidality of stationary sinusoids and is not suitable for modulated sinusoids. As highlighted by several authors, it still remains very difficult to decompose the overlapping partials into their original sources, even if the number of concurrent sources is known beforehand [H. Viste and G. Evangelista, 2002, Virtanen, 2003, Every and Szymanski, 2004] and [Yeh and Roebel, 2009]).

Beating

Quasi-periodic signals have several different periods, as mentioned earlier. This means that several periods have competitive fitness to the signal, this may cause some ambiguity in what concerns to determine the correct F0. Besides that, with partials the same happens too, which results in different partials with slight frequency deviations, changing over time. Therefore, if two overlapping partials with similar amplitude suddenly have a small frequency difference, **beats** can be observed (see Figure 2.28).

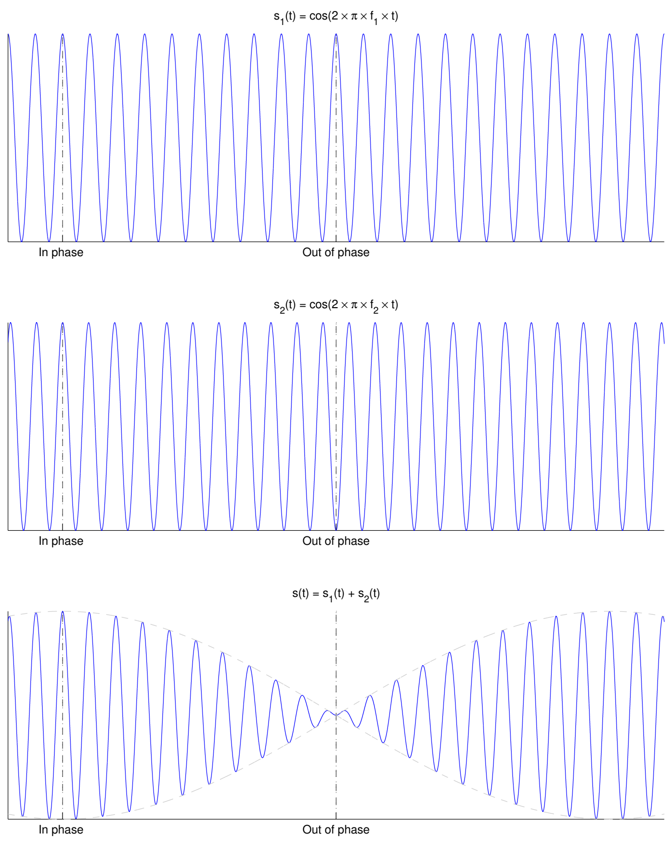


Figure 2.28: Interference tones of two sinusoidal signals of close frequencies f_1 and f_2.

[Wood, 2007] explained the dissonance as a physical phenomenon, where it is heard unpleasant beats. As depicted in Figure 2.28, beats are periodic variations of loudness,

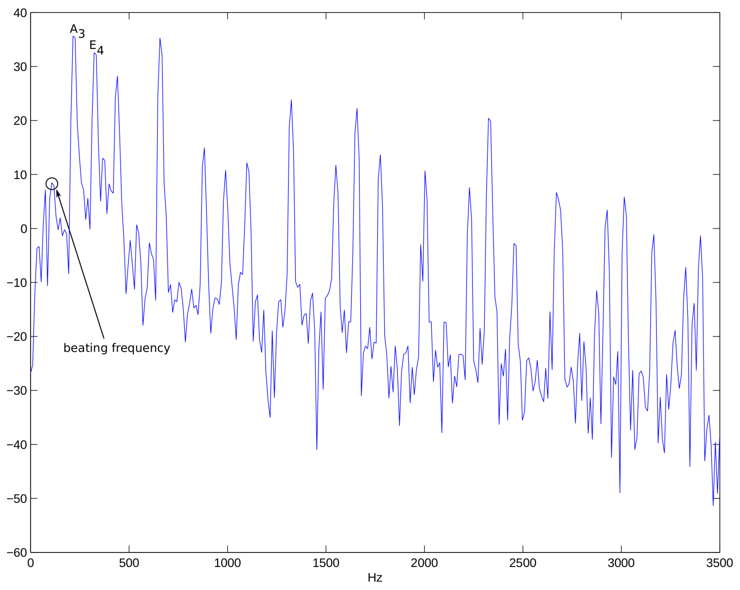


Figure 2.29: Example spectrum of two piano sounds with fundamental frequencies A3 (220.0 Hz) and E4 (329.6276 Hz). A beating component appears at frequency 110 Hz, corresponding to a A2 ghost pitch. Adapted from [Reis, 2012]

and the frequency of the beats depend on the frequency difference of the two tones:

$$s(t) = s_1(t) + s_2(t) = \cos(2\pi f_1 t) + \cos(2\pi f_2 t) \tag{2.54}$$

which may be simplified using trigonometry:

$$s(t) = 2\cos\left(2\pi\frac{\Delta f}{2}t\right)\cos\left(2\pi\frac{f_1+f_2}{2}t\right), \tag{2.55}$$

being $\Delta f = f_1 - f_2$, resulting in a single sinusoidal wave, with frequency $\frac{f_1+f_2}{2}$ and with a periodic variation of loudness whose frequency is Δf. This phenomenon introduces spectral components that are not present in any original source, producing "ghost" fundamental frequencies and, also, changing the original partial amplitudes across the spectrum (see Figure 2.29).

Physical properties of instruments

Commonly, music is a combination of musical sounds produced by more than one instrument. Each instrument has specific spectral characteristics. This diversity increase the complexity of the problem of multiple-F0 estimation.

1. **Spectral envelopes**: It is the shape or contour of the signal spectral principal peaks, which are in general the partials. Many musical instruments produce sounds with smooth spectral envelopes however with different shapes, Figure 2.30, [Yeh, 2008a]).

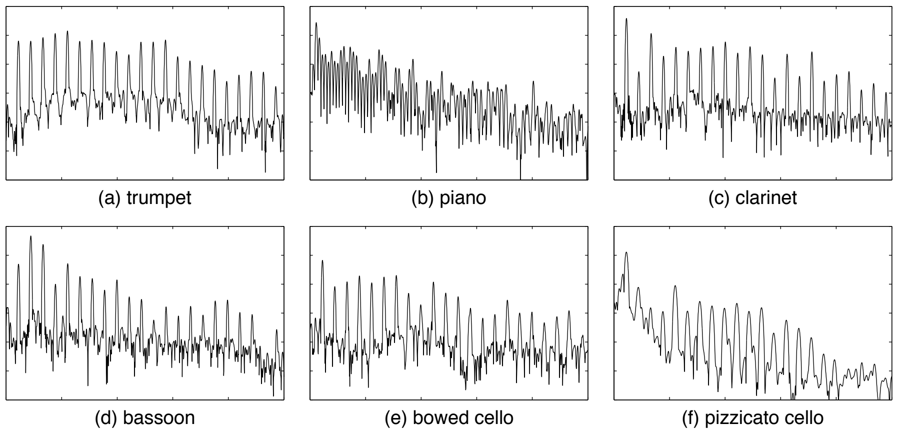


Figure 2.30: The spectra of six musical instrument sounds: (**a**) trumpet A3 note; (**b**) piano A3 note; (**c**) clarinet A3 note; (**d**) bassoon A3 note; (**e**) bowed cello A3 note; and (**f**) pizzicato cello A3 note. ([Yeh, 2008a], page 17).

2. **Inharmonic partials**: An ideal harmonic sound, has harmonics with frequencies multiples of the fundamental frequency F0, although the partials of real musical instruments do not have frequencies exactly integral multiple of the fundamental frequency. The frequencies of the partials, for stretched strings, follow the equation:

$$f_h = hF\sqrt{1 + \beta\left(h^2 - 1\right)}, \tag{2.56}$$

in which F is the fundamental frequency, h represents the harmonic number, and β is the inharmonicity factor ([Fletcher and Rossing, 1998], page 363).

3. **Spurious components**: There are some dominant components excited along with the partials in some instruments. According to[Conklin Jr, 1999], **phantom partials** are usually present in string instruments, appearing close to the frequencies of the normal partials. In some cases phantom partials become more dominant then the normal usual partials.

Transients

The **transient** definition is not precise however it can be considered as an event or zone of short duration where a fast variation of the signal happens [Rodet and Jaillet, 2001]. On music signals, transients are identified as note onsets in the form of fast attacks, or at note offsets with fast releases. Fundamental frequency estimation is a very difficult task when applied in sound zones with transients.

Transients are typically impulsive and covered with high energy, resulting in many spurious components that will most likely interfere with other sound sources. Some researches consider the transient as a signal component, where the transient is detected by a non-parametric approach [Rodet and Jaillet, 2001, Bello et al., 2005, Röbel, 2003], or a parametric approach [Daudet, 2004, Molla and Torrésani, 2004].

Reverberation

Reverberation prolongs sounds in time generating overlapping with the subsequent sounds. Therefore, a music sound signal is composed by not only by the direct sound part but also by the reflected part and the reverberated part. Therefore, even considering a monophonic sound produced by an instrument, if it is in a reverberant environment the recorded sound looks like a polyphonic [Beauchamp et al., 2012, Yeh et al., 2006]. Furthermore, the reverberated parts are not stationary, thus it turns the process of analysing a sound signal even more complex.

2.8 Summary

This chapter introduced briefly some important background topics, from waves and signal sampling and periodicity to more complex concepts on signal processing. After that some theoretical concepts were addressed related with frequency estimation of sound signals.

Chapter 3

Related Work

This chapter deals with related work on Multiple Pitch Estimation of piano sounds. It presents a brief history of the approaches to the problem. Several techniques are presented following a classification proposal. The main features and techniques used to tackle the MPE problem are briefly explained. The strengths and weaknesses of other approaches are described and discussed. It is also justified the decision of using Cartesian Genetic Programming to the problem, as well as the main advantages and features that are pursued.

Multiple fundamental frequency estimation was first studied by [Shields, 1970] during is research on separating co-channel speech signals, a few years later this problem topic was extended to polyphonic pitch estimation pursuing the goal of automatic music transcription for polyphonic sounds, by [Moorer, 1977] and [Piszczalski and Galler, 1977].

There has been a lot of research on Multiple Pitch Estimation over the years, however at this time there is no agreed general-purpose system or method. If we consider a more generic task such as AMT, the several approaches can be generally organized into two main categories, [Benetos et al., 2018]: *note-level* and *frame-level.* Note-level transcription, or note tracking, is one level higher than MPE, in terms of the richness of structures of the estimates. It not only estimates the pitches in each time frame, but also connects pitch estimates over time into notes. Frame-level transcription, or Multi-Pitch Estimation (MPE), is the estimation of the number and pitch of notes that are simultaneously present in each time frame, this is usually performed in each frame independently.

There are several classifications proposals, according to [Yeh, 2008b] MPE approaches in the context of AMT can be categorized into joint estimation and iterative estimation. The approaches categorized as iterative estimations are based on finding successive F0s: They search for the predominant F0 and cancel it, then start a new iteration to find another fundamental frequency until a stopping criterion. The cancellation process is used to remove the F0, its harmonics and also subharmonics, otherwise it would infer noise and could lead to false results on future detection processes. These approaches tend to be light in what concerns to computational cost, however the iteration process accumulates errors. The Joint estimation techniques tend to consume more computational resources. Instead of evaluating one fundamental frequency at each iteration, they evaluate F0 combinations, typically increasing the results in terms of accuracy.

[Su and Yang, 2015] presented another type of classification for multi-pitch estimation, which depends on the use or not of labeled training data-set with ground-truth pitches. Most of the researches done earlier used unsupervised learning, although with the recent developments in machine learning systems several new approaches depend on supervised learning. To fill that need, the researchers have a few available labeled datasets of which [Emiya et al., 2010], [Goto et al., 2002], [Fritts, 2006] stand out.

[Benetos et al., 2013] states that recent developments in AMT show that the vast majority of proposed approaches now falls within the "joint" category. Thus, it purposes a classification model that organises multi-pitch detection systems according to the core techniques or models employed: feature-based, statistical model-based or spectrogram factorisation-based. Featured-based techniques do not use a specific model, but try to devise measures of pitch salience and criteria for selecting and scoring pitch candidates from time-frequency representations, these techniques apply methods based on signal processing. Statistical model-based techniques use probabilistic methods to model the spectral peaks or envelops. Spectrogram factorisation-based techniques use templates of spectral patterns of different pitch combinations and then decompose an input magnitude spectrogram according to the activation of different templates. In the rest of this chapter it will be used this classification model extended with an additional emergent class, neural networks, to present and describe some of the most important approaches to the MPE problem, regardless some subjective issues related with some technique precise classification.

3.1 Feature-based

Typically, feature-based methods use some audio features derived from the time-frequency representation of polyphonic music to estimate multiple pitches in a joint or iterative fashion.

3.1.1 Hypothetical Partial Sequence

[Yeh et al., 2010], [Yeh, 2008b] present a well succeeded frame-based approach for estimating single-channel polyphonic music signals using short time Fourier transform (STFT) representation. The system classifies the spectral peaks into sinusoids and noise with an adaptive noise level estimation. The Rayleigh distribution is used to model the spectral magnitude distribution of noise [Yeh and Roebel, 2006]. The plausibility of a set of F0 hypotheses is jointly evaluated, in order to match as many sinusoidal peaks as possible, taking into consideration the overlapping partials. For each hypothesis, the frequencies and the amplitudes of their **hypothetical partial sequences** (HPS) are calculated by partial selection, using a harmonic matching technique and an overlapping partial treatment. The joint estimation algorithm is based on the characteristics of harmonic instrument sounds: harmonicity, the smoothness of spectral envelope and synchronous evolution of partial amplitudes. The score function is a linear combination of four criteria: harmonicity (HAR), mean bandwidth (MBW), spectral centroid (SPC) and the standard deviation of mean time (SYNC). The HAR criterion evaluates the harmonic matching between the combination of the HPS and the observed spectral peaks. The MBW criterion evaluates the frequency of the envelope of a HPS by its bandwidth. The SPC criterion is used to prevent subharmonic errors. The SYNC criterion estimates the mean time for each individual peak, in order to evaluate the synchronicity of the temporal evolution of the partials in a HPS. These criteria are then combined by the sum of the peak salience of the related partials. An iterative approach is used for candidate selection. The candidate solution method starts with non-harmonically related F0 candidates, that explain the biggest number of sinusoidal peaks, and then, the harmonically related F0 candidates within the previous F0s found. A F0 hypothesis is considered a valid estimate if it either explains significant energy or improves the spectral smoothness of the set of the valid F0 estimates.

3.1.2 Cancellation by Spectral Models

[Klapuri, 2003] presented an iterative estimation approach based on spectral smoothness and harmonicity. The input signal is preprocessed by a RASTA-like technique [Hynek Hermansky, 1993] on a logarithmic frequency scale in a way that the spectral magnitudes are compressed and the additive noise is removed. The preprocessed spectrum is split and organized in frequency bands. At each sub-band, F0 weights are calculated by normalizing the sum of their partial amplitudes. Those weights are then combined taking inharmonicity into account. The predominant F0 source is smoothed and subtracted from the signal spectrum in order to avoid its corruption after multiple iterations of direct cancellation. After the subtraction, the overlapping partials still persist in the remaining sources. The method described uses the average amplitude within one octave band in order to smooth out the envelope of an extracted source, and is called the **band-wise smooth model**. The process repeats itself by computing the weights of each candidate and extracting the F0 candidate until the maximum weight related to signal-to-noise ratio (SNR) is below a fixed threshold. A perceptually motivated multiple-F0 estimation method is presented by [Klapuri, 2005]. The input signal is split into multiple frequency bands by using a bank of bandpass filters, which models the frequency selectivity of the inner ear. Each sub-band signals are compressed, half-wave rectified and low-pass filtered. The magnitude spectra is summed across channels and used to perform the harmonic matching to extract the predominant F0. A $1/k$ **smooth model** is used to remove the predominant source from the mixture, keeping the energy of higher partials for the next iterations. [Klapuri, 2006] presents a spectral model which attempts to generalize a variety of musical instrument sounds. An input signal spectrum is flattened to suppress timbral information. Then, the salience of a F0 candidate is calculated as a weighted sum of the amplitudes of its harmonic partials. [Santoro and Cheng, 2009] present an algorithm for multiple F0 estimation in the transform domain, based on Klapuri's work, to function in the Modified Discrete Cosine Transform (MDCT) domain. Klapuri's work still be considered an important milestone in MPE, and used in benchmarks.

3.1.3 Combined Frequency and Time Domains

[Peeters, 2006] and [Emiya et al., 2007b] use both frequency and lag domain features to tackle the problem of single-pitch estimation. They multiply a spectral and a temporal representation of the input audio signal to determine the likelihood of a pitch candidate. [Bello et al., 2006] extended this idea to multi-pitch estimation of piano music and proposes a system based on a hybrid method, where the frequency domain approach is associated to a time–domain process. This method takes into account the information contained in phase relationships that are lost when only the magnitude spectra of sounds are analyzed. [Su and Yang, 2015] extends [Peeters, 2006] and [Emiya et al., 2007b] work for multi-pitch estimation and propose an unsupervised feature-based approach referred to as Combined Frequency and Periodicity. This system detects pitches according to both harmonic series in the frequency domain and a subharmonic series in the quefrency domain. The log-scaled amplitude spectrum is used for the frequency representation of the signal, which is then pseudo-whitened to spectrally flatten the signal as in [Klapuri, 2003]. The generalized cepstrum is employed for the temporal representation of the signal. After thresholding both signals, a peak picking process is applied to all local maxima and discards other non-peak terms. The presence of a true pitch is identified by three conditions: a prominent harmonic series in the frequency representation, a prominent subharmonic series in the temporal representation and the fundamental frequency of the harmonic series and the fundamental period of the subharmonic series match at the same fundamental frequency. Criteria to deal with missing fundamental frequencies and stacked harmonics are presented. False positives are reduced by sparcity contraints. In post-processing, pitches above C5 that leave any other pitches in the affinity of 0.1 seconds by more than one octave are discarded. Fiaally, a post-processing filtering operation is done in neighbor frames for temporal smoothness which connects non-continuous estimates and removes isolated notes shorter than 0.12 seconds.

3.1.4 Blackboard Systems

A blackboard system is an artificial intelligence approach based on the blackboard architectural model, designed to handle complex problems, where a common knowledge base, the "blackboard", is iteratively updated by a diverse group of specialist knowledge sources, starting with a problem specification and ending with a solution [Nii, 1986b]. According to [Nii, 1986a], a blackboard-system application consists of three major components: knowledge sources, blackboard and control shell. The knowledge sources are a diverse group of specialists, each one being a self-cointained expert on some aspects of the problem which can contribute to the solution independently of the particular mix of other specialists [Corkill, 1991]. The blackboard is a common knowledge base, shared repository of problems, partial solutions, suggestions, and contributed information. It is constantly being updated by the knowledge sources, in order to achieve a solution. The control shell is responsible for controlling the flow of problem-solving activity in the system, organizing the common knowledge sources in the most effective way. [Martin, 1996] presents a blackboard approach to automatic music transcription where a new system is proposed, based on the log-lag correlogram [Ellis, 1996]. [Bello and Sandler, 2000], [Bello et al., 2000] present a system based on a top-down approach. The blackboard system is composed of three hierarchical levels. The inputs are the result of a segmentation routine in the form of an averaged STFT matrix. The blackboard has a hypotheses database, a scheduler and knowledge sources. One of those sources is a neural network, trained for chord recognition, which allows the system to output more than one note hypothesis at a time.

3.1.5 Multi-spectral analysis

Argenti et al. proposed a polyphonic music transcription method based on the bispectrum [Argenti et al., 2010], where the cross-correlation between the two-dimensional (2-D) harmonic template and the bispectrum is calculated to estimate the frame-wise pitches. Through this method, the over- lapping harmonics from different notes are mapped into different locations in the 2-D plane. However, it is difficult to recognize the pattern if there are few harmonics or some partials are missing, and the template for the bispectrum of harmonic signals is a bit complex.

To address this problem, [Zhang et al., 2020] proposed a pseudo 2-D spectrum-based method: the pseudo 2-D spectrum is first constructed to map the time domain signal into the 2-D frequency space, where the harmonic signal exhibits a typical 2-D pattern. Then, pitch estimation is carried out by cross-correlation between the pseudo 2-D spectrum and the fixed 2-D harmonic template

3.2 Statistical Model-Based

Statistical model-based methods formulate the multi-pitch estimation problem in a statistical framework, then maximum a posteriori (MAP) or maximum likelihood (ML) estimation are employed to select the most salient pitch in each iteration

3.2.1 Maximum a Posteriori Estimation Approach

[Emiya et al., 2007a] address the problem of single-pitch estimation with a technique based on a Weighted Maximum Likelihood principle. The signal is decomposed into a sum of sinusoidal components and a colored noise. A moving average process is assumed for the noise while the spectral envelope of the partials is modeled by an autoregressive model. The fundamental frequency is calculated by following a Weighted Maximum Likelihood principle, which simultaneously whitens both noise and sinusoidal sub-spectrums. This technique is extented for multi-pitch estimation, by jointly evaluate multiple F0's at the same time. [Emiya, 2007] incorporates an onset detector presented by [Alonso et al., 2005]. For each segment, the first frames are analised and the largest peaks will result in a set of F0 candidates. Then, for each frame and for each combination of notes among the selected candidates, the likelihood of the spectrum is derived, according to the previous work in [Emiya et al., 2007a]. The maximum likelihood estimation is embedded into a Hidden Markov Model framework. Finally, detected pitches in consecutive frames and segments are merged together. [Emiya et al., 2009] extend this approach to multipitch estimation of multiple concurrent pitches in piano sounds. A new spectral model is employed where the inharmonic distribution is taken into account and adjusted for each possible note. A smooth autoregressive model is introduced to model the spectral envelope of the overtones and a low-order moving-average (MA) process is used for the residual noise. [Goto, 2004] propose a method called *PreFEst* to estimate the most predominant F0 of melody and bass lines in audio signals. Maximum A Posteriori Probability estimation is employed to represent every possible F0 as a probability density function, by using the expectation-maximisation algorithm [Dempster et al., 1977]. [Kameoka et al., 2007] present a multipitch analyzer called the harmonic temporal structured clustering method. This method jointly estimates multiple fundamental frequencies, onsets, offsets and dynamics. The harmonic structure model is an extension of the [Goto, 2004] work. The frequency of each partial is modelled using a Gaussian distribution function and the spectra is obtained by the constant Q transform. The synchronous evolution of partials is modelled by Gaussian mixtures.

3.2.2 Time-domain Bayesian

[Davy et al., 2006] extended the polyphonic time-domain Bayesian harmonic model presented in [Walmsley et al., 1998], [Walmsley et al., 1999], [Davy and Godsill, 2003] and [Godsill and Davy, 2002], for multi-pitch transcription, with some modifications. The authors use a Gabor representation of nonstationary signals and a Markov chain Monte Carlo method for the parameter estimation algorithm. The model supports time-varying amplitudes and inharmonicity. [Peeling and Godsill, 2011] propose a new method for solving the problem of multi-pitch estimation using novel statistical models. The partial frequencies in the frequency domain are modeled by an inhomogeneous Poisson process. [Koretz and Tabrikian, 2011] addresses the problem of multi-pitch estimation using a combination of the maximum likelihood and maximum a posteriori probability criteria. Each of the fundamental frequencies is modeled by a Markov process. The dominant signal is modeled as a harmonic source and the remaining sources are modeled as Gaussian interference. The dominant source is estimated and removed from the mixture, and the process is applied to the next harmonic source.

3.2.3 Maximum-Likelihood Approach

[Duan et al., 2010] presents a maximum likelihood approach to multiple fundamental frequency (F0) estimation. The audio signal is splitted into multiple frames. To each frame, the Short Time Fourier Transform, hamming window and zero-padding are applied, to obtain a power spectrum. Spectral peaks are detected using a peak finder presented in [Duan et al., 2008]. The parameters for the model are learned from monophonic and polyphonic data. The system models the power spectral density as both spectral peaks and non-peak regions. The peak likelihood aims to find F0s that have harmonics that explain peaks and the non-peak likelihood aims to avoid F0s that have harmonics in non-peak regions. [Yoshii and Goto, 2012] present a statistical method called infinite latent harmonic allocation (iLHA) to deal with multiple fundamental frequencies (F0s) estimation in polyphonic music audio signals. The method relies on hierarchical nonparametric Bayesian models that can deal with complex models of multiple F0's and their harmonic structures.

3.3 Spectrogram factorization-based

The majority of recent multi-pitch estimation works use and expand spectrogram factorisation techniques or neural networks[Benetos et al., 2018]. Spectrogram factorization-based methods attempt to decompose the spectrogram of the audio mixture into a linear combination of notes with their corresponding intensities or probabilities. The recordings are processed at the music-piece level, and the model parameters are often estimated using expectation maximization or non-negative matrix factorization-like algorithms.

3.3.1 Non-negative Matrix Factorization

Non-negative Matrix Factorization (NMF) is a technique for data analysis where a non-negative matrix $\mathbf{V}$ is factorized into two non-negative matrices $\mathbf{W}$ and $\mathbf{H}$: given n $n \times m$ non-negative matrix $\mathbf{V}$ and a positive integer $r < \min(n, m)$, NMF tries to factorize $\mathbf{V}$ into an $n \times r$ non-negative matrix $\mathbf{W}$ and an $r \times m$ non-negative matrix $\mathbf{H}$ such that:

$$\mathbf{V} \approx \mathbf{WH}. \tag{3.1}$$

This way, the observed power spectra $\mathbf{V}$ can be decomposed into spectral models (basis functions in $\mathbf{H}$) of each note with its intensity change along time (weightings in $\mathbf{W}$) as in [Smaragdis and Brown, 2003]. The cost function is designed to favor the minimization of the residual with specific constraints like sparseness [Cont, 2006].

Sparse approximation or sparse decomposition accounts for most or all information of a signal with a linear combination of a small number of elementary signals called atoms, chosen from a dictionary. The technique of finding a representation with a small number of significant coefficients is often referred to as **sparse coding**. Decoding merely requires the summation of the relevant atoms, appropriately weighted.

Consider the observed signal x, where $x = D\alpha$. D is an $m \times p$ matrix ($m \ll p$) and $x \in \mathbb{R}^m, \alpha \in \mathbb{R}^p$. D is the *dictionary* or the design matrix. The idea is to estimate the signal α, subject to the constraint that it is sparse. **Sparsity** implies that only a few components of x are non-zero and the rest are zero. This implies that x can be decomposed as a linear combination of only a few $m \times 1$ vectors in D, called *atoms*: the basis of x. The sparse decomposition problem is represented as:

$$\min_{\alpha \in \mathbb{R}^p} ||\alpha||_0 \quad \text{such that} \quad x = D\alpha, \tag{3.2}$$

where $||\alpha||_0 = \#\{i : \alpha_i \neq 0, i = 1, \cdots, p\}$ is a pseudo-norm, l_0, which counts the number of non-zero components of $\alpha = [\alpha_1, \cdots, \alpha_p]^T$. This problem is NP-Hard with a reduction to NP-complete subset selection problems in combinatorial optimization. A convex relaxation of the problem can instead be obtained by taking the l_1 norm instead of the l_0 norm, where

$$||\alpha||_1 = \sum_{i=1}^{p} |\alpha_i|. \tag{3.3}$$

The l_1 norm induces sparsity under certain conditions [Donoho, 2006].

[Lee et al., 2010] assume that the Fourier coefficients of an input frame are a linear combination of the Fourier coefficients of previously recorded waveforms of individual piano notes [Bello et al., 2002] and defined a matrix of Fourier coefficients of segments of those waveforms as the dictionary. The computational complexity of the problem is reduced by l_1 minimization.

A modified sparse Non-negative Matrix Factorization algorithm is used for realtime pitch observation. [Vincent et al., 2010] and [Bertin et al., 2010] present a NMF in a Bayesian framework applied to polyphonic music transcription, which uses a model of superim- posed Gaussian components. The likelihood function incorporates spectral smoothness constraints. A space-alternating generalized expectation-maximization (SAGE) algorithm is applied to estimate the parameters . [Ochiai et al., 2012] proposes a NMF for estimating simultaneously basis spectra and activations, detecting note onsets and duration, and determining beat locations.

[Benetos and Dixon, 2011] proposed a model that extends the **shift-invariant probabilistic latent component** (PLCA) method of [Smaragdis et al., 2008]. This model is able to support the use of multiple pitch templates extracted from multiple sources. Using a log-frequency representation and frequency shifting, detection of notes that are non-ideally tuned, or that are produced by instruments that exhibit frequency modulations is made possible. Sparsity is also enforced in the model, in order to further constrain the transcription result and the instrument contribution in the production of pitches. Finally, a hidden Markov model-based note tracking method is employed in order to provide a smooth piano-roll transcription. A similar approach is used by [Arora and Behera, 2015] where a scheme is presented for the source transcription of pitched polyphonic music. Firstly, the multi-F0 values are extracted using source-filter model based Probabilistic Latent Component Analysis (PLCA) with harmonic dictionaries Secondly, the unsupervised source clustering scheme clusters the F0s into source specific clusters using Hidden Markov Random Field (HMRF) model.

3.3.2 Evolutionary Algorithms

[Reis and Vega, 2007], [Reis et al., 2007] consider music transcription as a search problem and present a new method for multi-pitch estimation on piano recordings, using genetic algorithms. Although the authors show the feasibility of the approach, the genetic algorithm tends to create additional notes in harmonic locations of the original notes to minimize the timbre differences between the original audio signal, and the internal samples. [Reis et al., 2012] presents a method which consists in a genetic algorithm aided by two major components: an adaptative spectral envelope modeling and a dynamic noise level estimation. The system process begins with an onset detector, which splits the input time signal into multiple frames. For each frame, a genetic algorithm, is triggered to perform the transcription. After applying the genetic algorithm, all the segments are joined into one whole transcription, and an Hill-Climber algorithm is used to merge consecutive notes. Each individual represents a candidate transcription.An adaptative threshold component encodes the dynamic noise level estimation, by adjusting the noise level for each frequency bin in the spectrum. The spectral envelope component encodes the internal synthesizer, by using the gain of its harmonics, expressed in dB, and its inharmonicity deviation for each partial, in order to best match the input piano in the original signal. A general system to encode the harmonic deviation of each partial was adopted to work with other kinds of pitched instruments. Each solution is first rendered by an internal synthesizer. Then, the fitness function, based on the log spectral distance, compares the input audio signal with the generated transcription, in the frequency domain.

3.4 Neural Networks

As for many tasks relating to pattern recognition or classification problems, neural networks (NNs) have had a considerable impact in recent years on the problem of music transcription and on music signal processing in general. The significant evolutionary step for neural networks took place in 1999, when computers started becoming faster at processing data and graphics processing units (GPU) were developed. Faster processing, with GPUs processing pictures, increased computational speeds by 1000 times over a 10 year span. During this time, neural networks began to compete with support vector machines. Neural networks also have the advantage of continuing to improve as more training data is added. The term Deep Learning became common in scientific language in the field of machine learning, deeper networks with more layers and able to train with huge datasets. NNs are able to learn a nonlinear function (or a composition of functions) from input to output via an optimization algorithm such as stochastic gradient descent [Goodfellow et al., 2016]. Compared to other fields including image processing, progress on NNs for music transcription has been slower but is now rising.

One of the earliest approaches based on neural networks was Marolt's Sonic system. A central component in this approach was the use of time-delay (TD) networks, which resemble convolutional networks in the time direction, and were employed to analyse the output of adaptive oscillators, in order to track and group partials in the output of a gammatone filterbank. Although it was initially published in 2001, the approach remains competitive and still appears in comparisons in more recent publications [Ewert and Sandler, 2016]. In the context of the more recent revival of neural networks, a first successful system was presented by [Böck and Schedl, 2012]. One of the core ideas was to use two spectrograms as input to enable the network to exploit both a high time accuracy (when estimating the note onset position) and a high frequency resolution (when disentangling notes in the lower frequency range). This input is processed using one (or more) Long Short-Term Memory (LSTM) layers.

[Sigtia et al., 2016] focus on long-range dependencies in music by combining an acoustic front-end with a symbolic- level module resembling a language model as used in speech processing. Using information obtained from MIDI files, a recurrent network is trained to predict the active notes in the next time frame given the past. This approach needs to learn and represent a very large joint probability distribution, i.e., a probability for every possible combination of active and inactive notes across time – note that even in a single frame there are 288 possible combinations of notes on a piano. To render the problem of modeling such an enormous probability space tractable, the approach employs a specific neural net- work architecture (NADE), which represents a large joint as a long product of conditional probabilities – an approach quite similar to the idea popularized recently by the well-known WaveNet architecture.

[Kelz et al., 2016] focus on the acoustic modeling and report on the results of a larger scale hyperparameter search and describe the influence of individual system components. Trained using this careful and extensive procedure the resulting model outperforms existing models by a reasonable margin. In other words, language models have led to a drastic improvement in performance. He extended is work i in [Kelz et al., 2019] using a late-fusion approach to piano transcription, combined with a strong temporal prior in the form of a handcrafted Hidden Markov Model (HMM). The network architecture under consideration is compact in terms of its number of parameters and easy to train with gradient descent. The network outputs are fused over time in the final stage to obtain note segmentation, with an HMM whose transition probabilities are chosen based on a model of attack, decay, sustain, release (ADSR) envelopes, commonly used for sound synthesis. The note segments are then subject to a final binary decision rule to reject too weak note segment hypotheses.

Also recently, Google Brain team presented a new method based on Deep learning [Hawthorne et al., 2017]. Combining and extending ideas from existing methods, this approach combines two networks: one network is used to detect note onsets and its output is used to inform a second network, which focuses on detecting note lengths

3.5 Discussion

As stated before, besides the several attempts using many different techniques and even distinct classification proposals, Multi Pitch Estimation remains an unsolved problem, and it still is the focus of many works and researches. The majority of multiple-F0 estimation and multi-pitch estimation methods rely on the frequency domain representation. This representation has the advantage of represent the harmonic structure and the spectral envelope of a sound in an intuitive way. Thus it eases the modeling of sound sources. Frequency domain representation or spectral can be done using multi-resolution or fixed-resolution.

Multi-resolution techniques, as filter-banks constant-Q transform or wavelets, have the advantage of representing the signal with different resolutions for distinct frequency bands. Although, some authors claim that using a multi-resolution spectral representation results in a benefit due its similarity to the equal tempered music scale or the human auditory system [Marolt, 2004, Klapuri, 2005] no physical evidences were presented to sustain that multi-resolution is better for representing the structure of a harmonic sound. [Hainsworth, 2003] and [Yeh, 2008a], claim that multi-resolution representations do not solve the time-frequency main issue and, the multi-resolution advantage becomes also a disadvantage: wavelets sacrifice frequency resolution at high frequencies, which can be a major drawback for distinguishing individual partials of concurrent sources, and constant-Q and wavelets loose temporal precision in the lower frequencies [Reis, 2012]. Therefore, most of the MPE approaches use fixed resolution and during this thesis the fixed representation is used through STFT.

Statistical methods can model varying time-frequency envelopes like, for instance, saxophone sounds. However some of those methods, like the Bayesian waveform models, are mathematically complex and have high computational cost. Some featured-based models like Blackboard systems are usually, general architectures and these systems depend on other methods to use them as knowledge sources. Spectogram factorization-based methods usually have a high computational cost and are also time consuming. Some of them depend on dictionaries with templates. There is also the problem of using harmonicity rules for a certain kind of music, typically western music. The methods based on Deep Learning with neural networks are in vogue, they have accomplished relevant accuracy results for AMT. However these approaches need specific and expensive hardware to train the networks as well as huge detests. Another important question related with some Machine Learning based methods is the fact that they enclosure a black box, part of the system works but it is hard to find out how and also how to tune it. At the end of the day, when working they cannot be decoded into a function or a mathematical process well explained.

There is a lack of approaches based on Evolutionary Algorithms for this known problem, the work of [Reis, 2012] is one exception. Among Evolutionary Algorithms available flavors, Genetic Programming is a very well known and developed area with some prominent works [Gandomi et al., 2015], and as far as it is of public knowledge, there are no approaches to multi pitch estimation. Cartesian Genetic Programming as a form of Genetic Programming has already prove to be able to solve classification problems particularly in image processing [Miller and Harding, 2008, Harding et al., 2013]. It is also a relative new methodology. Because all of this it was decided to face the challenge of implementing from scratch a system based on cartesian genetic programming to tackle the problem of multi pitch estimation, which is described in this dissertation. In a certain way, it is a pioneer innovation, the majority of CGP applications are related with other kind of problems specially image classification or segmentation. It may be also a prof of concept: to prove that CGP is a valid technique for music and sound processing problems. Another important advantage of CGP systems is the resulting classifier or set of classifiers transparency: it is possible to see, understand, export or tune the resulting expression for the classifiers and this features are important for educational and didactic purposes. Therefore, it may be considered a white box system that allows optimization. There are few systems that work real-time, thus another goal of the approach of this thesis is real-time processing based on a parallelizable architecture.

Chapter 4

Evolutionary algorithms and Cartesian Genetic Programming

Computing optimal solutions for many problems of relevant scientific importance is often difficult and sometimes impossible. Unlike exact methods, metaheuristics allow solving hard and complex problem instances by delivering satisfactory solutions in a reasonable time. All metaheuristic search methods try to solve problems for which no reasonable fast algorithms have been developed, and they are especially fit for optimization problems [de Vega, 2001].

Evolutionary computation is the most studied population-based metaheuristic; it uses stochastic algorithms, which use simulated evolution as a search strategy to iteratively evolve candidate solutions, using operators inspired by genetics and natural selection. The Evolutionary Computation (EC) is based on Darwin's theory of Evolution [Darwin, 1859] and in Mendel's laws of inheritance [Mendel and Bateson, 1925]. The species evolution is based on the survival of the fittest, they have more chance to pass on their genetic inheritance through breeding and consequently the progressive disappearance of the lees fit individuals. There are 3 important cornerstones:

- **Outbreed** - Will only survive a part (μ) of the descendants ($\mu + \lambda$);
- **Variability** - There is variability in the whole species in what concerns to structure and corporal function;
- **Inheritance** - Parents will pass to their descendants may of their own characteristics through genetic transmission.

Evolutionary Algorithms or EAs are population-based metaheuristics inspired by genetics and natural selection. The natural evolution process occurs in a evolving way over a population. The populations is a set of reproductive individuals with inheritance, each individual is scored with a fitness which influences its probability of breeding. Hence, to ensure evolution the individuals should have the ability to reproduce, their chance of surviving depends on their characteristics, that may also change due to mutations. The individual characteristics pass from parents to children due to inheritance and the entire population competes for limited and shared resources.

Some mechanism's characteristics are fundamental for the evolutionary process:

- **Evolution** - operates more over the chromosomes than over the structures of life encoded on them;
- **Natural selection** - is the linkage between the chromosomes and behaviors of their decoded structures;
- **Reproduction** - is the process where evolution turns visible.

The biological evolution process is future oriented, the past does not interfere. Any evolved specie extinguishes if its population fails to adapt. The fittest means the most adapted to the conditions and environment. The fittest will survive and perpetuate their adapted born characteristics to the next generations. Computer science classifies this as an optimization process and uses it as an advantage to solve difficult problems trough the Evolutionary Algorithms. EAs include some distinct paradigms and classes, however all of them should possess the following attributes [Michalewicz, 1994]:

- a genetic representation for potential solutions to the problem;
- a mechanism to create an initial population of potential solutions;
- an evaluation function that simulates the role of the environment, scoring solutions in terms of their "fitness";
- genetic operators that promote variety and children composition;
- values for the algorithm parameters (e.g., probabilities, population size).

Evolutionary Algorithms begin with an initial set of individuals, that represent candidate solutions for a specific problem, then with the aid of some search operators they refine that set using some gradient follow techniques based on biological mechanisms of solution space explorations and genetic operators [Rudolph, 1998, Oliveto and Witt, 2012], thus EAs are solution generation methods. There are important methods used on EAs that allow evolution or are important tools for genetic operators: Evaluation is responsible for rate an individual fitness, and selection allows choose some individuals to breed. Traditional breeding operators include Reproduction, Crossover, and Mutation [Holland, 1975]:

- **Reproduction** is the process of copying individuals. They are chosen according to their fitness value;
- **Crossover** is the procedure of mating the members of the new population, in order to create a new set of individuals. As genetic material is being combined, new genotypes will be produced;
- **Mutation** modifies the values of one or several genes of an individual.

Evolutionary Algorithms are based on the notion of competition. They maintain a population of solutions rather than just one current solution; in consequence, the search is afforded many starting points, and the chance to sample more of the search space than local searches. The *population* is iteratively recombined with *crossover* and mutated to evolve successive *populations*, known as *generation*. Various *selection* mechanisms can be used to decide which individuals should be used to create offspring for the next generation; key to this is the concept of the *fitness* of individuals.

The way evolutionary algorithms work on a problem is described in Algorithm 1. It as a peculiar lexicon and terminology analogue to genetic biology terms represented in italic in the last paragraph. It is usual to the term *chromosome* for an individual solution or sometime *genotype*, [Miller, 2011], Chapter 1.

Algorithm 1 Generic Evolutionary Algorithm

```
1:  Generate initial population size $p$
2:  Set generation number $g = 0$
3:  **repeat**
4:      Calculate the fitness of each individual of population;
5:      Select number of Parents according to quality;
6:      Recombine some parents to create offspring chromosomes;
7:      Mutate some parents and offspring;
8:      Form new population from mutated parents and offspring;
9:      *Optional* - promote a number of unaltered parents from step 5 to new population;
10:     $g \leftarrow g + 1$;
11: **until** ($g = maxgenerations$) **OR** (a fitness value criterion is reached)
```

The evolutionary algorithm starts by generating a number of chromosomes or individuals randomly; this is called the initial population (**step 1**). Thus, after this step we would have a number of individuals that encode solutions. The next step is to evaluate each individual genotype in the population. This is referred to as determining the fitness of the members of the population (**step 4**). The next step is to select the members of the population that will be used to generate the new population of potential solutions (**step 5**). Often these chromosomes are called *parents* as they are used to create new individuals called *children*. Usually, children are generated using *recombination* or *crossover*. A child is generated from two parents, by selecting genes from each one (**step 6**). The next step in the generic evolutionary algorithm (**step 7**) is to mutate some parents and offspring. **Step 8** of the evolutionary algorithm forms the new population from some combination of parents, offspring and their mutated counterparts. In the optional **step 9**, some parents are promoted to the next generation without change. The algorithm continues attractively until a fitness criterion is reached or a maximum number of iterations (generations).

4.1 Evolutionary Algorithms Classification

Using the EAs for solving problems imposes certain conditions on the nature of the problem. First of all, multiple solutions of the problem must exist, and the evaluation function must be provided; it is necessary for comparing the solutions and for directing the population towards areas of a solution space where optima are possibly located. The second essential condition is the ability to encode a solution within individual's genotype and to define the genetic operators, which modify the genotype and introduce a random factor to the whole procedure. The classical evolutionary algorithms can generally be grouped into four categories: genetic algorithms, genetic programming, evolutionary programming and evolution strategies [Michalewicz, 1994], (see Figure 4.1):

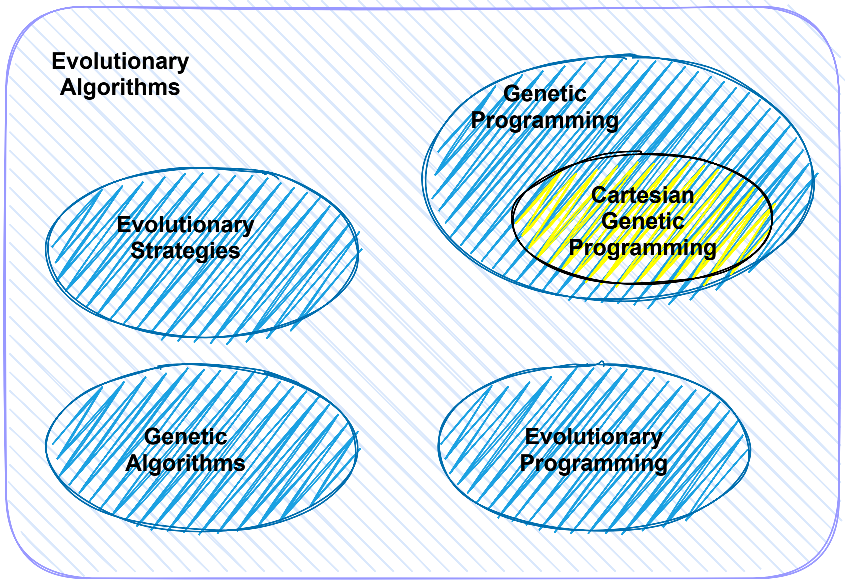


Figure 4.1: Classification of Evolutionary Algorithms (EAs)

- **Genetic Algorithms** (GA) which use genetic operators over chromosomes and the concept of mating and recombination. GAs were introduced by [Holland, 1975] and were first used for optimization problems by [De Jong, 1975];
- **Evolutionary Strategies** (ES) which emphasize the behavior changes among individuals. ES were introduced by [Rechenberg, 1973] and had some significant developments with [Schwefel, 1975];

- **Evolutionary Programming** (EP) which focuses on behavioural changes among species. It was first introduced by [Fogel et al., 1966] and designed by [Fogel, 1992] in its currently practiced form;
- **Genetic Programming** (GP) which uses evolution to evolve programs or mathematical expressions capable of solve a given problem. The individual is a program or a mathematical function. GP applies the genetic operators to programs or mathematical expressions represented as trees of operands and operators [Koza, 1992, Koza, 1994].

A typical evolutionary algorithm ($\mu + \lambda$) [Rechenberg, 1973] also considered a classical Evolutionary Strategy [Rutkowski, 2008] is widely used among severall flavours of Evolutionary algorithms. Algorithm 2 shows its implementation in pseudo-code.

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

```
1: $t \leftarrow 0$;
2: Initialize $P_0$ with $\mu$ individuals chosen uniformly at random;
3: repeat
4:     for i = 1 to $\lambda$ do
5:         choose $x_i \in P_t$ uniformly at random;
6:         mutate each gene of $x_i$ with probability $p$;
7:     end for
8:     Create the new population $P_{t+1}$ by choosing the best $\mu$ individuals out of $P_t \cup \{x_1, \ldots, x_\lambda\}$;
9:     $t \leftarrow t + 1$;
10: until a stop condition is fulfilled.
```

It starts with a population of μ individuals and then generates λ offspring individuals. The next generation is composed by the fittest μ individuals from the previous generation of $\mu + \lambda$ individuals. The idea is create a temporary population; it has the different size than the parent population (depending on the assumed parameters μ and λ). In this step, the fitness values are not important. Individuals in the temporary population undergo crossover and mutations or just mutation for our particular case. From such populations, an assumed number of the best individuals are selected to the next generation of the population (in a deterministic way). This evolutionary strategy is the general case for the most common evolutionary strategy used in CGP that makes ($\mu = 1$)and will be detailed later on this chapter.

4.2 Genetic Programming

Genetic Programming (GP) is considered a Genetic Algorithm. GP is a specialization of GAs usually associated with the evolution of tree structures; it focuses on automatically creating computer programs by means of evolution [Koza, 1992].

In order to optimise a computer program, the notion of suboptimal programs instead of right or wrong programs must be considered [Luke, 2011]; Therefore, GP is usually

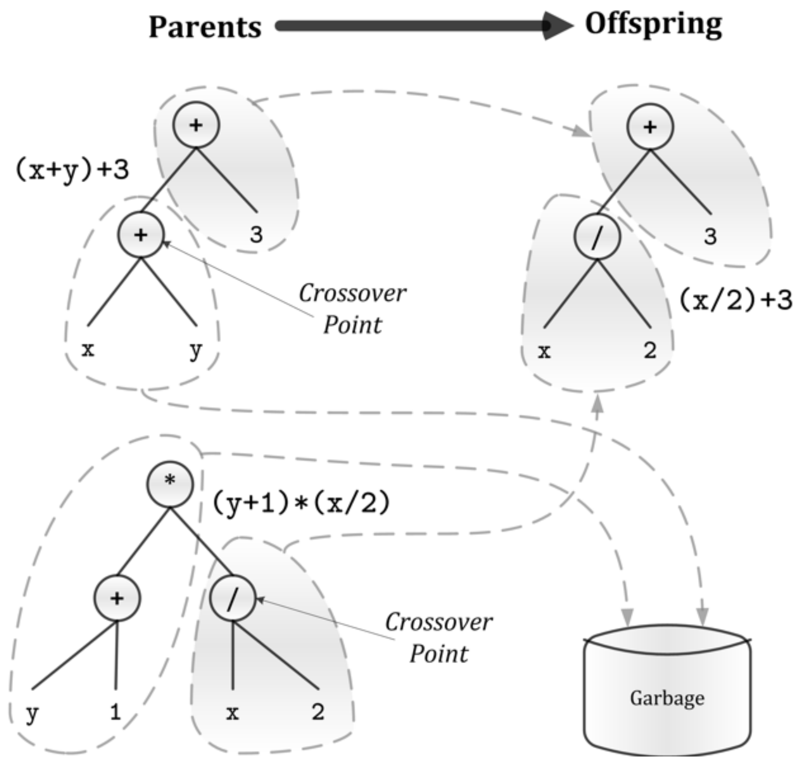


Figure 4.2: Example of GP subtree crossover. Adapted from [Poli et al., 2008]

concerned in the space of possible programs, but it is not clear which ones outperform the others and in what degree. In most GP approaches, the programs are represented using variable-sized tree genomes. The leaf nodes are called terminals, whereas the non-leaf nodes are called non-terminals (or functions). Terminals can be inputs to the program, constants or functions with no arguments; non-terminals are functions taking at least one argument. The definition of the terminals and non-terminals depends on the target application. GP uses a Function-set that is the set of functions from which the GP system can choose when constructing trees. GP builds new trees by repeatedly selecting nodes from a function Set and by arranging them. The individuals in the initial population are typically randomly generated.

One important unique and distinguished feature of GP among the Evolutionary Algorithms flavours is the implementation process for Crossover and Mutation [Poli et al., 2008]. The Subtree Crossover method, depicted in Figure 4.2 is the most commonly used: given two parents, Subtree Crossover randomly selects a crossover point (i.e., a node) in each parent tree; then, the offspring is created by replacing the subtree rooted at the crossover point in a copy of the first parent with a copy of the subtree rooted at the crossover point in the second parent. The most commonly used form of Mutation in GP is Subtree Mutation (Figure 4.3), which randomly selects a mutation point in a tree and substitutes the subtree rooted there with a randomly generated subtree. Another common form of mutation is Point Mutation: a random node is selected and the primitive stored there

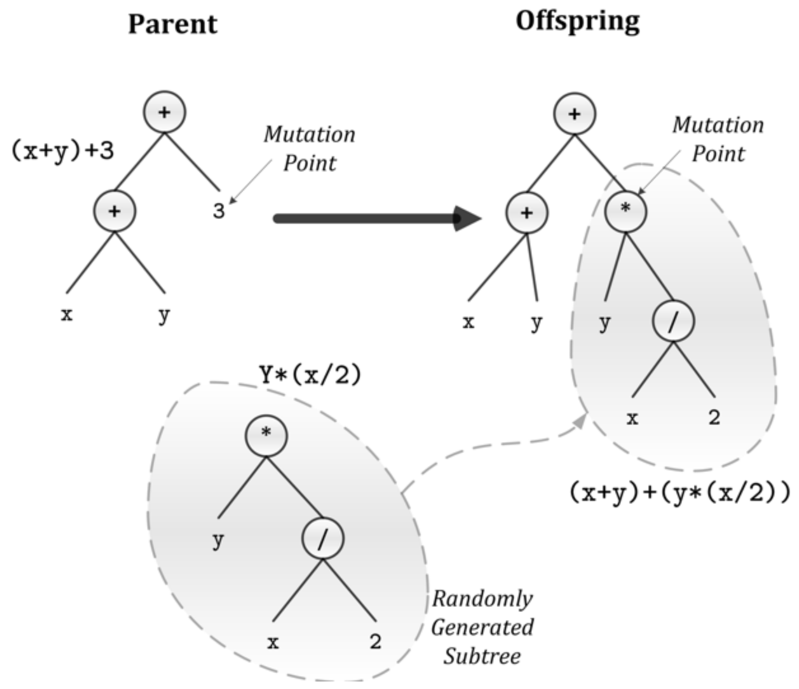


Figure 4.3: Example of GP subtree Mutation. Adapted from [Poli et al., 2008]

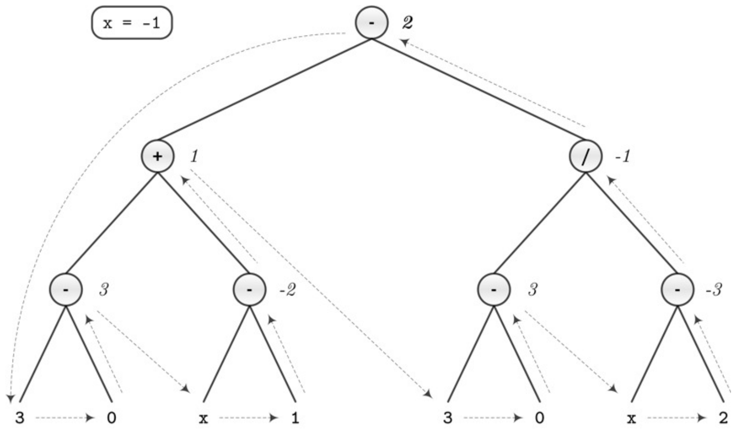


Figure 4.4: Example interpretation of a GP syntax tree (the terminal x is a variable and has a value of -1). The number to the right of each internal node represents the result of evaluating the subtree rooted at that node. Adapted from [Poli et al., 2008]

is replaced with a different random primitive of the same arity taken from the primitive set. As well as in the GAs the reproduction operator is based in selection of an individual according to fitness values, and then a copy of it is produced and passed to the next generation.

To evaluate computer programs using a fitness function, the most common and natural way is is to execute them and assess their behaviour [Luke, 2011]]. Decoding a program tree usually means executing tree nodes in an correct order, that guarantees that nodes are not executed before the value of their arguments (if any) is known. This is usually performed by traversing the tree recursively starting from the root node, and postponing the evaluation of each node until the values of its children (arguments) are known [Poli et al., 2008]; this crucial process is graphically detailed in Figure 4.4. The specification of the control parameters in a run is a mandatory preparatory step. There are several parameters; some of the most important include the population size, the probabilities of performing the genetic operations, the minimum and maximum tree sizes, and the stopping criteria. It is impossible to define general guidelines for setting optimal parameter values, as these depend greatly on the details of the application. However, GP is in practice robust, and it is likely that many different parameter values will work [Poli et al., 2008].

4.3 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) roots come out of the work of [Miller et al., 1997], as a method of evolving digital circuits. However, the term "Cartesian Genetic Programmin" appeared two years later in [Miller, 1999]. Cartesian Genetic Programming is an increasingly popular and efficient form of Genetic Programming [Koza, 1992, Koza, 1994] proposed by Julian Miller in 2000 [Miller and Thomson, 2000]

It is being applied to many fields, such as machine learning, neural networks, data mining, financial prediction, function optimization, classification, electronic circuit design, and so on. According to [Miller, 1999], CGP is more efficient than standard GP methods in learning Boolean functions. CGP is "Cartesian" because it encodes programs as a two-dimensional grid of nodes that are addressed in the **Cartesian** coordinate system (see Section 4.3.2). In its classic form, it uses a very simple integer based genetic representation of a program in the form of a directed graph instead of a tree. Graphs are very useful program representations, more general than trees, and can be applied to many domains (e.g. electronic circuits, neural networks). Therefore CGP is now considered as a general form of Genetic Programming [Miller and Thomson, 2000] and it has been and increasingly popular and efficient form of Evolutionary Algorithms. It is seen as a flexible form of Genetic Programming that is concerned with the automatic evolution of computational structures, such as computer programs, mathematical equations or functions. CGP encodes a graph representation of computer programs. The computational structures encoded are represented as a string of integers (sometimes real values). These integers, known as genes, determine the functions of nodes in the graph, the connections between nodes, the connections to inputs and the locations in the graph where outputs are taken from. Therefore, the genotype consists of a list of integers (and possibly real parameters) that represent the program primitives and how they are connected together.

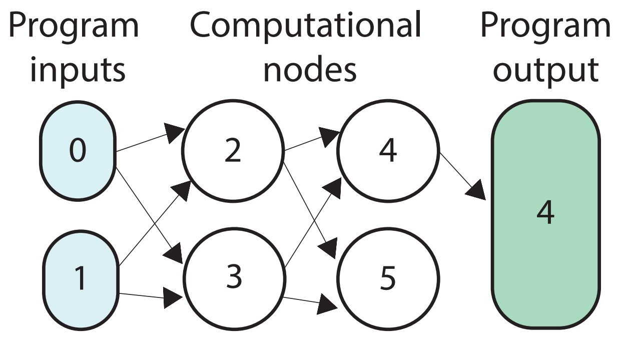


Figure 4.5: CGP program structure: Program inputs and computational nodes are numbered sequentially. The program outputs can link to any computational node or program input.

4.3.1 Program Structure

CGP evolves mathematical structures or programs capable of perform a task to optimize or solve a problem. The individuals are those structures. The CGP programs are graphs of nodes, where those nodes are structured in a Cartesian way: it can be seen as a matrix of nodes with a definened number of rows and columns. CGP Programs have three major components: program inputs, computational nodes and program outputs. **Computational nodes** are structures organized and composed by input connections and a function. The input connections of a node have their origin in any program input or other precedent nodes. The function is among the ones previously defined in a look-up table and it takes as arguments the values received through the node's inputs. The node itself is indexed by an integer value so that it can be referenced by other node input connections. The computational nodes, organized in a two-dimensional grid of nodes, are numbered sequentially and linked directly between them in a feed-forward manner (see Figure 4.5). A program can have several inputs, named **program inputs**. **Program outputs** are indexes that link to some nodes. For example, if the program's output is the number 4, the result of the program is the value computed by node 4's function (see Figure 4.5). Program inputs and nodes are referenced by sequential numbers.

In Figure 4.5 we can see that the program has two inputs, four nodes and one output. The grid of nodes that is the base for the CGP graph is defined by the *rows* (n_r) and *columns* (n_c). The number of inputs and outputs is not dependent of the grid (matrix) dimensions, it is only influenced by the problem characteristics, therefore depending on the n_r and n_c a wide range of grids can be made and consequently a wide range of graphs. Figure 4.6 illustrates 3 different examples of CGP structures. For a problem with 2 inputs and 4 outputs we may use many different matrix dimensions, in the upper case we have a grid of nodes with $n_r = 6$ and and $n_c = 3$, and in the middle one we use a matrix of nodes with $n_r = 3$ and and $n_c = 6$. A more generic and usual structure is also illustrated in the bottom of Figure 4.6 where a single row with 6 columns grid is used to address a problem with 4 inputs and 1 output. In general choosing a single row imposes the least constraints.

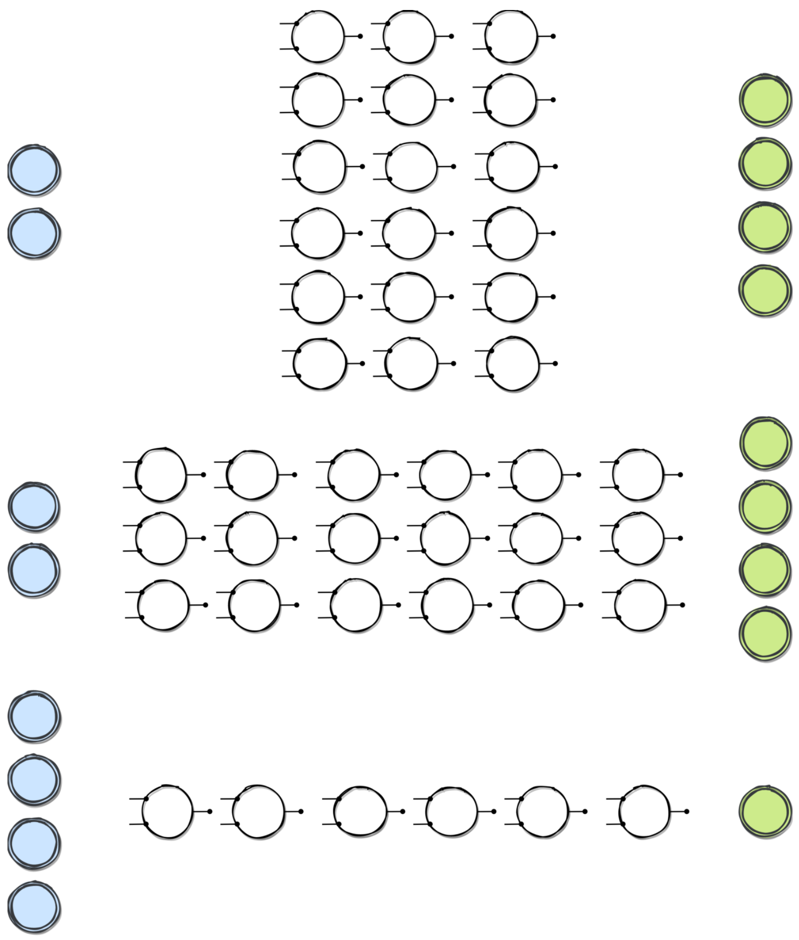


Figure 4.6: CGP nodes matrices: 3 different nodes matrices structures and different number of inputs and outputs, (blue) input nodes, (withe) computational nodes and (green) output nodes. (**top**) 2 inputs, 4 outputs, matrix $n_r = 6$, $n_c = 3$; (**midle**) 2 inputs, 4 outputs, matrix $n_r = 3$, $n_c = 6$; (**bottom**) 4 inputs, 2 outputs, matrix $n_r = 1$, $n_c = 6$

4.3.2 CGP general form

In CGP, programs are represented in the form of directed acyclic graphs. These graphs are represented as a two-dimensional grid (matrix) of computational nodes. The genes that make up the genotype in CGP are integers that represent where a node gets its data, what operations or function the node performs on the data and where the output data required by the user is to be obtained. In CGP, each node in the directed graph represents a particular function and is encoded by a number of genes. One gene is the address of the computational node function in the function look-up table. We call this a function gene. The remaining node genes say where the node gets its data from. These genes represent addresses in a data structure (typically an array). We call these connection genes. Nodes take their inputs in using a feed-forward manner from either the output of nodes in a previous column or from a program input (which is sometimes called a terminal). The number of connection genes of a node is the maximum function arity from the function-set (function look-up table). The program data inputs are given the absolute data addresses 0 to $n_i - 1$ where n_i is the number of program inputs. The data outputs of nodes in the genotype are given addresses sequentially, column by column, starting from n_i to $n_i + L_n - 1$, where L_n is the maximum number of nodes defined by the user. The general form of a Cartesian genetic program is shown in Figure 4.7, it is a grid of nodes whose functions are chosen from a set of primitive functions. The grid has n_c columns and n_r rows. The number of program inputs is n_i and the number of program outputs is n_o. Each node is assumed to take as many inputs as the maximum function arity a. Every data input and node output is labeled consecutively (starting at 0), which gives it a unique data address which specifies where the input data or node output value can be accessed (shown in the figure on the outputs of inputs and nodes) In general, there may be a number of output genes (O_i) which specify where the program outputs are taken from or eventually the output nodes are fixed as the last ones from the array. The graph is directed and feed-forward; this means that a node may only have its inputs connected to either input data or the output of a node in previous columns. The genotype structure is presented bellow the the CGP scheme in Figure 4.7.

All node function genes f_i are integer addresses in a look-up table of functions. All connection genes $C_{i,j}$ are data addresses and are integers values between 0 and the address of the node at the bottom of the previous column. CGP has three parameters that are chosen by the user. These are the the number of rows (n_c), the number of columns (n_c), and *levels-back* (l). The product of the first two parameters determine the maximum number of computational nodes allowed: $L_n = n_c \times n_r$. The parameter l controls the connectivity of the graph encoded. Levels-back constrains which columns a node can get its inputs from. For instance if $l = 2$, a node can only have its inputs connected to the outputs of any nodes in the immediate left two columns or a primary input. If one wishes to allow nodes to connect to any nodes on their left, then $l = n_c$. Varying these parameters can result in various kinds of graph topologies. Choosing levels-back to be one leads to highly layered graphs in which calculations are computed column by column. In contrast if the levels-back is chosen to be equal to the number of columns and number of rows is one we are facing a special and very common case of parameters values combination widely used where we face an array of nodes instead of a grid and a node can be connected to any other previous node.

4.3.3 Genotype

The **genotype** is the codification of a program as it is used and manipulated by the CGP algorithm. It describes what are the programs inputs, computational nodes and their functions, program outputs and how they are connected together. In general, it is a list of genes where each gene is an integer with a particular meaning. As we have seen earlier, program inputs and nodes are referenced by their index. Since a node is a structure with input connections and a function, each node has multiple genes. Taking the example

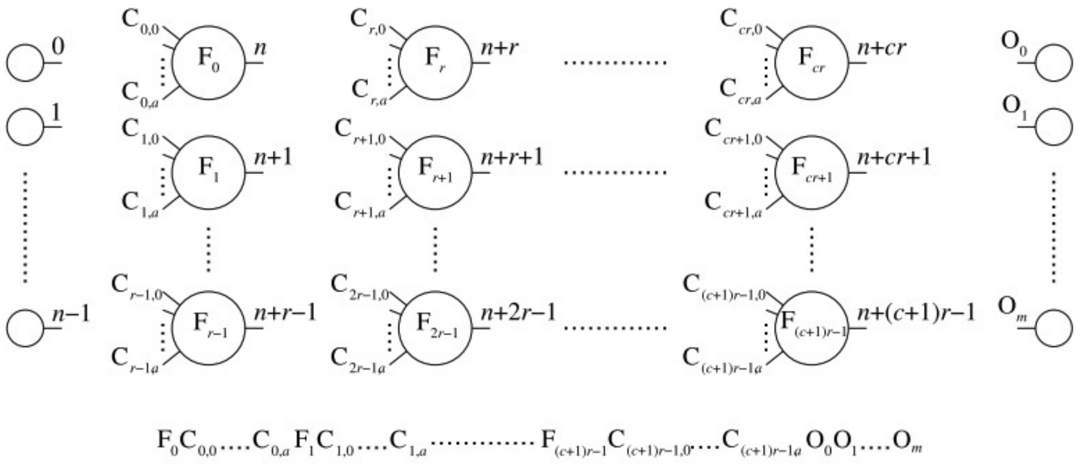

$F_0 C_{0,0} \dots C_{0,a} F_1 C_{1,0} \dots C_{1,a} \dots F_{(c+1)r-1} C_{(c+1)r-1,0} \dots C_{(c+1)r-1a} O_0 O_1 \dots O_m$

Figure 4.7: General form of CGP: A grid of nodes with n_c columns and n_r rows, adapted from [Miller, 2011]

depicted in Figure 4.8, it illustrates the CGP encoded list of genes (genotype) and the correspondent graph of nodes (phenotype) for an academic problem were the individuals are represented as a set of mathematical equations. The CGP system uses 2 inputs (x_0, x_1) and 4 outputs (O_A,O_B,O_C,O_D) and the grid of nodes has 6 nodes distributed by 2 rows and 3 columns. The genetic structure that encodes a node consists in 3 genes with diverse meanings: first references the function value and then the values of the node's connections (Figure 4.9). The node indexed with number 2 contains 3 genes with the following sequence (0 0 1): the first gene corresponds to the address of the function in the lookup table or the function index, the second and third genes correspond to the node inputs, also known as nodes connections. For the particular case of the node represented in Figure 4.9, extracted from the graph of the Figure 4.8 we may observe the node and its correspondent genetic representation as well as the resulting mathematical expression after decoding, the node function is a sum of the node 0 and node 1 ($x_0 + x_1$), which are system inputs, and the node index is 2, the first after the inputs. Usually, all functions have as many inputs as the maximum function arity and unused connections are ignored. This introduces an additional redundancy into the genome.

In Figure 4.8 we have the entire genotype with 6 computational nodes from index 2 to 7 and with 4 output nodes, the index 0 and 1 are reserved for the inputs. We also have the function set represented in a small table with 4 functions indexed from 0 to 4 (+, -, *, /). The phenotype is represented by the resulting graph also illustrated. The node number 6 is in dashed line and it is represented in the genotyoe however it is not used for computing the outputs so it is not used when decoding the phenotype. That node and its genes are usually called **non-coding** or **silent genes**.

The final result of the CGP program is obtained decoding the phenotype graph, in this particular example the 4 outputs are obtained in the form of mathematical functions of the 2 inputs:

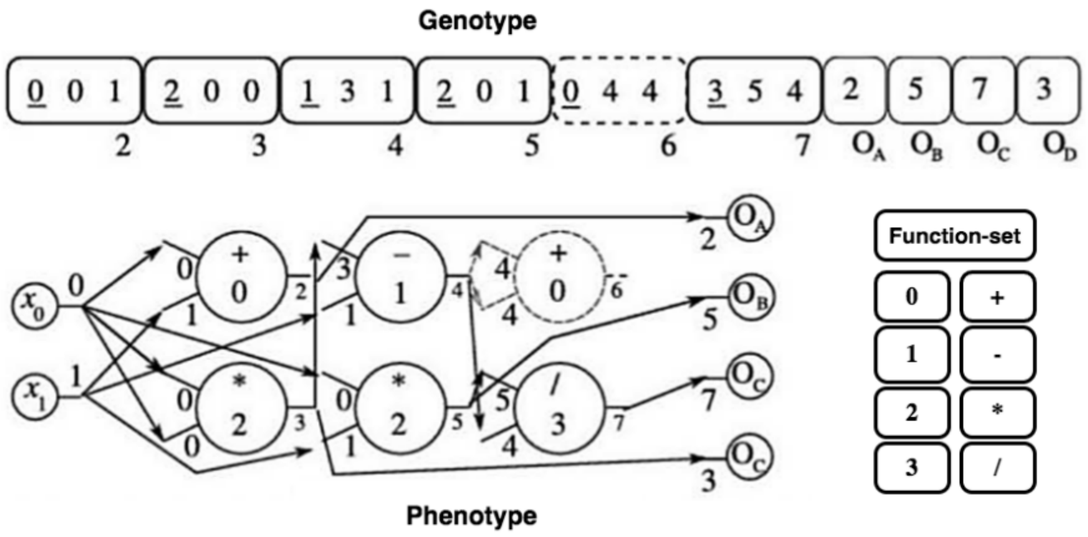


Figure 4.8: A CGP generic example of a genotype, its corresponding phenotype and the function set. There are 2 inputs (x_0, x_1) and 4 outputs (O_A,O_B,O_C,O_D). $n_c = 3$ columns and $n_r = 2$ rows.

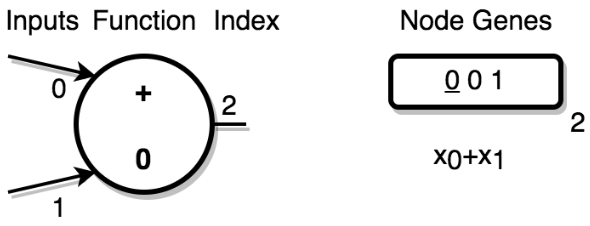


Figure 4.9: Computational node number 2: node with 2 connections, nodes 0 and 1. It will compute the function number 0 of the function-set. Node genes (0 0 1) and its mathematical meaning.

$$\begin{aligned} O_A &= x_0 + x_1 \\ O_B &= x_0 * x_1 \\ O_C &= \frac{x_0 * x_1}{{x_0}^2 - x_1} \\ O_D &= {x_0}^2 \end{aligned} \tag{4.1}$$

Thus, the set of Equations 4.1 is the resulting CGP program, obtained from the phenotype in the graph form that was encoded by the list of genes, genotype.

4.3.4 Allelic Constrains

The values that genes can take (i.e. alleles) are highly constrained in CGP. When genotypes are initialized or mutated, these constraints should be obeyed. There are some allelic constrains, that the genotype must respect. The alleles (values) of function genes f_i must take valid address values in the look-up table of primitive functions. Let n_f represent the number of allowed functions. Then f_i must obey to the following range:

$$0 \leq f_i \leq n_f. \tag{4.2}$$

There is another parameter called **levels-back** l, which determines how many previous columns of nodes may connect to a node in the current column. When $n_r = 1$ and $l = n_c$, any node can have input connections coming from any program input and any node on its left, which allows unrestricted connectivity. However, if $n_r > 1$, nodes cannot connect to other nodes in the same column. Then, having a node in column j, and $j \geq l$, node connections, C_{ij}, must obey to the following range:

$$n_i + (j - l)n_r \leq C_{ij} \leq n_i + j \times n_r. \tag{4.3}$$

If $j < l$, then the following condition must be met:

$$0 \leq C_{ij} \leq n_i + j \times n_r. \tag{4.4}$$

Program output genes O_i can connect to any node or program input:

$$0 \leq O_i < n_i + L_n, \tag{4.5}$$

where Ln is the number of nodes in the genotype, computed by the following:

$$L_n = n_r \times n_c. \tag{4.6}$$

This representation is very simple, flexible and convenient for many problems.

4.3.5 Genotype-Phenotype

One of the key characteristics of CGP is the genotype-phenotype mapping. The genotype is of fixed-length but the phenotype is not, duo to the fact that the genotype can have inactive genes. Thus, they are redundant because they cannot influence the programs output. The corresponding genes are called non-coding genes or inactive genes. This means that we can have a phenotype different from the genotype because non-coding genes are not expressed in the phenotype, that is, the program that will run in practice. For instance, the example of the CPG program represented in Figure 4.8 has a resulting genotype (**001 200 131 201 044 354 2 5 7 3**) whereas the phenotype is (**001 200 131 201 354 2 5 7 3**) i.e. a three genes shorter corresponding to one inactive node. The output or outputs of the CGP are nodes that point to other nodes (connection genes) and so on. Decoding the program is recursive in nature and works from the program output genes first. To decode the program outputs, the active nodes should be identified. The process begins by looking at which nodes are directly connected to the output genes. Then these nodes are examined to find out which nodes are directly linked to them. Since non-coding genes are not addressed, they present little computational overhead. The non-coding genes presented in the genotype are treated in a special way by the usual CGP algorithms, silent mutation is allowed to perform neutral search.

4.3.6 Evolutionary Strategy

The evolutionary strategy widely used for CGP is a special case of the $\mu + \lambda$ evolutionary strategy [Hansen et al., 2015] described in Algorithm 2 where $\mu = 1$ (Algorithm 3). This means that, in this special case, the population size is always one. At each iteration (generation), λ new offspring are generated from the current one through mutation. Then, the best among the current individual and the offspring becomes the new current individual in the next iteration. It is important to emphasize the condition in (step 10), the idea is to promote genetic variety and also silent mutation. If the current individual (parent) suffers a mutation in a silent node or gene, that mutation does not causes changes in the phenotype, (it only changes the genotype), thus the resulting mutate individual has exactly the same fitness of its parent, however it has a different genotype (silent mutation). Promoting this individual instead of its parent allows what is called neutral search, and it has been proved very importatant for CGP evolution and results[Miller, 2011]. Hence, 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.

Algorithm 3 $(1 + \lambda)$ CGP Evolutionary Strategy

```
1: for $i = 0$ to $\lambda$ do
2:     Generate individual $i$ randomly;
3: end for
4: Choose the fittest individual and promote it to next generation;
5: while solution not found AND max number of generations not reached do
6:     for $j = 1$ to $\lambda$ do
7:         Mutate parent to generate offspring $j$
8:     end for
9:     Choose the fittest individual according to:
10:    if An offspring individual has better or equal fitness than the parent then
11:        Offspring individual is select as fittest
12:    else The parent is select as fittest
13:    end if
14: end while
```

4.3.7 Mutation

The mutation operator used in CGP is a point mutation operator and it is very simple to implement. In a point mutation, an allele at a randomly chosen gene location is changed to another valid random value. If a function gene is chosen for mutation, then a valid value is the address of any function in the function set, whereas if an input gene is chosen for mutation, then a valid value is the address of the output of any previous node in the genotype or of any program input. Also, a valid value for a program output gene is the address of the output of any node in the genotype or the address of a program input. There is also the real parameters mutation when used as genes. This is a special case were the mutation process has to ensure that the new parameter value belongs to workable valu, usually we interval normalization is used. The mutation process can be managed in two distinct ways: a number of genes in the genotype that can be mutated in a single application of the mutation operator is defined by the user, and is normally a percentage of the total number of genes in the genotype; just define a mutation probability (p) for each gene in the genotype without concerns about the number of genes that can mutate.

Other genetic operators may be used in CGP, a fundamental one is **selection**, the selection operator is mainly expressed in steps 10, 11 and 12 of Algorithm 3, where the best among the current individual and the λ offspring is chosen as the next iteration individual. The **Crossover** operators don't usually receive much attention in CGP. In [Miller and Thomson, 2000], a one-point crossover operator was used but they found it to be disruptive to the subgraphs within the chromosome, which affected the performance of CGP. However, lately some work by[Clegg et al., 2007] has investigated crossover in CGP (and GP in general). Their approach uses a floating-point crossover operator, similar to that found in evolutionary programming, and also adds an extra layer of encoding to the genotype, in which all genes are encoded as a floating-point number in the range [0,1].

4.4 Summary

This chapter presented an introduction about evolutionary algorithms and all its classes. Among all the Evolutionary algorithms, it was given special focus on cartesian genetic programming as the main technique used in this thesis. Some CGP topics and theoretical fundamentals were detailed, like: evolutionary strategy, main algorithm, encoding process etc. This is a fundamental chapter in order to understand the capabilities of CGP and how they were employed to tackle the MPE problem. Next chapters will focus on the tool developed to apply CGP to several signal processing problems, and in the use of the toolbox to the problem of MPE of piano sounds.

Chapter 5

Cartesian Genetic Programming Toolbox for Matlab - CGP4Matlab

Cartesian Genetic Programming (CGP) has already demonstrated its capabilities on synthesizing complex functions, extracting main features from images and performing image segmentation [Harding et al., 2013]. As one of ours fist task to use Cartesian Genetic Programming to the problem of pitch estimation of piano music we decided to create a brand new toolbox for Matlab [1] named CGP4Matlab. It should not only be a tool tackle this problem but also capable of address signal processing problems. Matlab was our first choice for two main reasons: there were not any toolboxes on this topic (CGP) for Matlab and Matlab has recognized capabilities in signal processing including sound and image processing with a lot of useful functions already implemented and assembled in other toolboxes. There was at the time, some CGP tools, frameworks or implementations for *Java* or in *C* as well as some applications (most of them are available at CGP home page created by the principal author Julian Miller at https://cartesiangp.com). Although there are a number of public domain genetic algorithm and genetic programming toolboxes for MATLAB, there are no toolboxes for cartesian genetic programming. CGP4Matlab was developed as a contribution to the community, providing a free toolbox that can be used and extended by other researchers, allowing them to benefit from MATLAB's great mathematical potential on audio and image processing. Also, with this toolbox, researchers that already work with genetic programming in MATLAB are now able to try the cartesian version of genetic programming. CGP4Matalb toolbox is generic and flexible enough to be applied to any kind of audio or image processing problems. It is completely free and available for download[2].

The idea was to have a highly flexible toolbox, configurable throughout parameters and function callbacks, which could be used to several type of problems. After testing the

[1]MATLAB is a programming and numeric computing platform used develop algorithms, and create models.

[2]https://github.com/tiagoinacio/CGP4Matlab.

toolbox with simple problems the goal is to use it to the problem of MPE. The toolbox's architecture will be introduced throughout this chapter as well as each component and main features. Finally, it is shown an example of a symbolic regression problem as an example of toolbox usage.

5.1 Architecture

The developed CGP toolbox is simple to understand and to use, it allows an easy and quick way for encoding a specific problem. The toolbox uses the classical structure of CGP. One of the most important features projected for the toolbox was that it was generic enough to help developers encoding from small to bigger problems. To accomplish that goal a few design decisions were made that will be explained next.

5.1.1 Overview

All the possible combinations of rows and columns are allowed, considering that $n_r > 0$ and $n_c > 0$. The allelic constrains are generated dynamically, depending on the cartesian grid of nodes representation and regarding that *Levels-back* was also taken into consideration. The toolbox is also prepared to use additional parameters in genothype, these parameters have no number restrictions. It can be used any fitness function provided by the user. The toolbox is ready to receive as many inputs as desired by the user, and they may be of any type and value. The number of outputs is also variable depending on users need and problem specifications. The function-set is provided by the user and the function look up table is generated automatically. Furthermore, there is a system of callbacks and the evolutionary algorithm (EA) used is the $1 + \lambda$ with some adaptations, referred previously in Section 4.3.6.

5.1.2 Evolutionary Algorithm

The evolutionary algorithm implemented is similar to the Evolutionary strategy $(1 + \lambda)$ with some peculiar adaptations for the particular case of CGP. The encoded EA is presented in Algorithm 4 using detailed pseudo-code. one of the most important goals is to ensure that the toolbox is as generic as possible, so a few parameters used in the evolutionary process were chosen to be configurable.

The number of **offspring** (λ) is defined by the user. This is useful because there can be some problems that require a small number of offspring and others that require a bigger number of offspring. The **mutation rate** (p) is also configurable. This is the mutation probability for each gene. The maximum number of **runs**, mr, and maximum number of **generations**, mg, are also required parameters. Finally, the last parameter is the maximum or minimum **fitness**, f, for a solution to be considered valid, depending if we want to maximize or minimize the fitness function. The EA needs to know when a candidate solution can be considered as a valid solution for the problem, in order to stop the evolutionary process.

Algorithm 4 Algorithm $((1 + \lambda)\, EA)$

```
 1: $g \leftarrow 0$;
 2: Set current individual $I_0$ as the best of $\lambda$ individuals created randomly;
 3: $F_g \leftarrow$ fitness of current individual $I_0$
 4: while $g < mg$ and $F_g < f$ do
 5:     for i = 1 to $\lambda$ do
 6:         Create a copy $x_i$ of current individual $I_g$;
 7:         Mutate each gene of $x_i$ with probability $p$;
 8:     end for
 9:     Set $I'_g$ as the best of $\{x_1, \ldots, x_\lambda\}$;
10:     if $I'_g$ is better or equal than $I_g$ then
11:         $I_{g+1} \leftarrow I'_g$
12:     else
13:         $I_{g+1} \leftarrow I_g$
14:     end if
15:     $F_{g+1} \leftarrow$ fitness of current individual $I_{g+1}$ ;
16:     $g \leftarrow g + 1$;
17: end while
```

5.1.3 Components

The toolbox is divided into several components. Each one has its purpose and special role. The first one is the **CGP** component. It exposes all the functionality to encode an application built on top of the toolbox. This component communicates with the **EA** and **Structure** components. The Structure is just an helper, which stores the positions of the genes according to the type of gene (connection, function, program output and parameter). The EA component is responsible for initializing the runs in the evolutionary algorithm. It starts with a certain number of **Offspring**, created by the **Genotype** component which, in turn, is composed by the **Connection**, **Functions**, **Outputs** and **Fitness** components. **Run** is connected to the **Generation** component, by executing it multiple times. In each generation, **Mutation** can occur, which will change the genotypes (using the Connection, Functions, Outputs and Fitness components). Figure 5.1 shows the overall structure of the toolbox's components. Each component will be briefly addressed next.

5.2 Implementation

The methodology used to implement the CGP4Matlab toolbox was object-oriented programming (OOP). It is a well known programming paradigm based on the concept of **objects**, which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). It was developed in Matlab (version R2016a)

The *CGP* class provides access to an API that lists all the features needed to encode a program. It is the "core" of the toolbox and its primary component. The *CGP* class lets

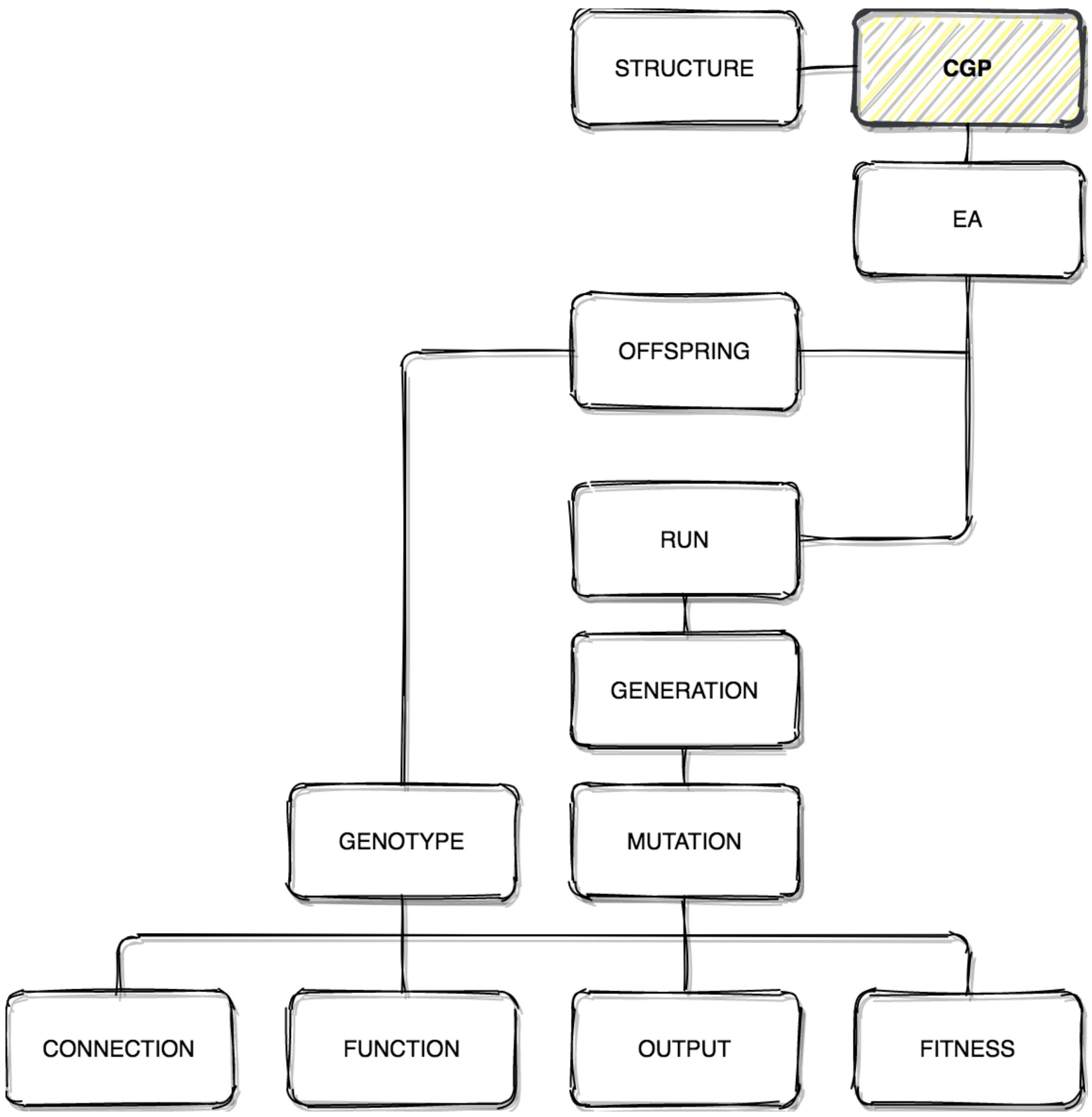


Figure 5.1: Toolbox components scheme

the user add the program inputs, provide the fitness function, add parameters and define the function-set. The constructor takes a configuration object. This object will contain all the configuration necessary for the CGP and for the EA.

For the CGP, the parameters are divided into: number of rows, number of columns, number of levels back and number of program outputs. Since some CGP approaches assume that the output node is the last node of the graph, this option was also taken into consideration. So, if we pass the value *last* to the *output_ type*, the last node of the genotype will be considered the program output. This option only works when the number of program outputs is set to 1, otherwise it will be ignored. Having these parameters configurable, the user has total control of the grid layout of the generated program (genotype). For the EA, the parameters are: maximum number of generations, maximum number of runs, number of offspring, mutation rate, the fitness threshold and the fitness operator. The fitness threshold is the limit for which a candidate solution is considered a valid solution to the problem. This allows the evolutionary process to stop or skip to the next run. In some kind of problems, the goal is to minimize an error rate, where 0 would be the best value for the fitness. Also, there are other problems where the goal is to maximimze the fitness function, as we have in our approach. The *fitness_solution* property covers that necessity. However, the operator to use in the comparison between fitness values also needs to be configurable, because the optimization of those values is different. The fitness operator, O, is the operator to use when comparing the new fitness candidate solution with the parent's fitness, and can take the following values: '>', '<', '>='and '<='. For example, consider the parent's fitness as f_0 and an offspring fitness as f_1: if $O =$ '>', $f_0 = 0.5$ and $f_1 = 0.6$, then the offspring will replace the parent in the new generation; if $O =$ '<', $f_0 = 0.5$ and $f_1 = 0.6$, the parent will remain as it have the best fitness. This operator is also used for checking if a solution is a valid solution for the given problem. Therefore, it is also used for comparison between a solution's fitness and the *fitness_solution* value, also configurable. Figure 5.1 describes every possible field for the configuration.

Table 5.1: Configuration table with the fields that the structure should have, the type of value and the description of each one.

Key	**Type**	**Description**
rows	double	number of rows
columns	double	number of columns
levels_back	double	number of levels-back
outputs	double	number of outputs
output_type	string	set the program output as the last node (last, random)
runs	double	number of runs
generations	double	number of generations
offspring	double	number of offspring
mutation	double	probability of mutation
fitness_solution	double	fitness for a solution to be considered valid
fitness_operator	string	fitness operator ('>','<','>=' or '<=')

At the time of instantiaton, the *CGP* class will verify if all the required settings were passed in the configuration object.

This class also exposes the functionallity of adding program inputs. Each problem requires a specific set of program input or inputs. Some may require one integer as input, others may require an array, or even a complex type of object. To address this abstraction, the input provided for the CGP toolbox is of type **struct** (structure). Each field in the structure is a program input. Therefore, the program inputs can be of any type: integers, strings, structs, arrays, matrix, etc. The number of fields present in the structure indicates the number of inputs that the toolbox needs to set in the genotype, which is dynamically set: there is no need to specify how many program inputs this program will have.

The fitness function is passed by callback (function pointer) to the program.

The toolbox reads the function set from a specific directory provided by the user. This directory should have all the functions that could be used in the genotype. All the functions should receive as many inputs as the maximum function arity. This is a requirement for the program to work. Besides the maximum function arity, if the user used added parameters to the genotype, these should be also passed to each function. This method will iterate through all the Matlab files in the directory passed as argument, and it will create a function handle for each one.

Some specific signal processing functions might require special arguments like ranges or constants to be executed (e.g.: a low pass filter needs to know which percentage of the original signal will be attenuated). Those parameters might need to evolve through time, because their best values for the contribution to the solution of the problem is unknown beforehand. The genotype can encode those parameters and add them to the evolutionary process. Parameters should have integer or double values. Each parameter is encoded by a structure with a name, a callback function for the initialization of the parameter value, and another callback function for mutating the value. The initialization and mutation functions should return an integer or a double. The mutation function should also accept an argument, that is the value of the current parameter to mutate. When running the algorithm, there are a number of events from the evolutionary process that can be useful to handle, for running additional scripts or simply to add some kind of report. In order to have that range of possibilities, the user is able to pass optional callbacks, each of which, will fire at the following events: the configuration has been set, a fittest solution is achieved after a run, a fittest solution is achieved in a generation, a new solution is created, a new generation starts, a new run starts and a genotype is mutated. After adding all the program inputs, fitness function, parameters and callbacks, the configuration callback is fired, with a few useful parameters about the configuration of the program. All the methods and properties of the *CGP* class are listed in figure 5.2.

There are several components that need to know how many genes are in the genotype, or if a specific gene is a function-gene or a connection gene. Instead of having to determine those properties multiple times and at different stages, this information is only computed once, in this class. The *Structure* class serves as an helper throughout the entire evolutionary process. The main goal is to classify each gene *a priori*, according to its type: connection, paremeter, program output or function. For example, if we have 3 genes per computational node and our genotype starts at number 1 (Matlab does not accept zero-based vectors), we know in advance that gene 1 will represent a function and genes 2 and 3 will both correspond to connections. Since this class will be responsible for defining the type of genes, it needs to know a few parameters, such as: the number of genes, the number of genes per node, the connection genes per node, the number of computational nodes and the number of parameters.

Figure 5.2: Properties and methods for the CGP class.

CGP
- config_ - callbacks_ - functions_ - inputs_ - parameters_

+ addCallbacks(callbacks : struct)
+ addFitnessFunction(fitness_function: function_handle)
+ addFunctionsFromPath(path: char array)
+ addInputs(inputs : struct)
+ addParameters(parameters : struct)
+ run()
- isValidConstructor_(configuration : struct)
- configuration_()
- areValidParameters_(parameters : struct)

The *EA* class is responsible for starting the evolutionary process. It iterates for the maximum number of runs, defined in the configuration of the CGP, storing the fittest candidate solution of each one.

The *Run* class is responsible for initializing a run. First, it generates a few candidate solutions. Then, it will start the evolutionary loop over the generations. The class stores the best candidate solution, while evaluating if a solution for the problem was found.

The *Generation* class is responsible for initializing a new generation. It starts with the previous fittest candidate solution (parent), and generates a few mutated versions, according to the configuration provided. If the λ chosen in the configuration phase is 4, it will generate four mutated versions of the parent solution. All the new genotypes are evaluated, and the fittest solution is stored.

The *Offspring* class is responsible for the initialization of a specific number of offspring, previously defined, at random, before iterating through the generations. It initializes randomly different genotypes which are then evaluated. The fittest solution is stored and used as the parent solution, for the generation loop initialization.

The *Genotype* class is responsible for the creation of a genotype, restricted to the configuration provided: number of columns, number of rows, number of program inputs, parameters, and so on. First, the function genes are added to the genotype. Then, the connection genes are randomly generated, as well as the parameters and program outputs.

After the genotype is created, the active nodes are recursively found by analysing the program outputs. For each output, the connection nodes are retrieved and stored in an array. For each of those, their connections are also saved in that array, and so on. This process stops until there are no more nodes to analyse. Lastly, the fitness of this new candidate solution is computed.

The *Connection* class is responsible for generating a random and valid connection for a specific node. It receives the connection gene index as argument. The class first finds which node belongs the connection gene. This is done by subtracting the number of program inputs from the gene index and dividing that value by the number of genes per node. Then, it finds all the possible connections for that node. This is achieved by recursively iterating through the previous nodes, taking into account that nodes in the same row cannot be connected between each other, and also taking into account the number of levels-back. Lastly, it randomly pick one connection from the possible connections.

The *Functions* class is responsible for randomly generating the function genes for the genotypes. It takes into account the number of functions present in the function-set, to be able to generate valid function genes. It can generate one function gene at a time or multiple function genes. This is is useful, because we find where all the function-genes are positionated in the genotype, and call this class once, which returns function genes to all those positions. If we have 10 nodes, we have to generate 10 function-genes in the genotype. If our function-set is composed by 5 functions, this class will generate 10 random values between 1 and 5, each corresponding to a function-gene mapped to one of the functions in the function-set.

The *Output* class is responsible for generating a valid program output. Depending on the settings provided initially, this class can pick the last node to be the program output, or randomly pick any program input or computational node in the genotype.

The *Fitness* class is responsible for calling the fitness callback provided in the configuration phase. A few properties are passed to that callback, such as the genes in the genotype, active nodes, function-set, program inputs and others. It has a validation of the type returned by the function, which should return an integer or double value. The returned value, is stored and used as the fitness of that particular candidate solution.

The *Mutation* class receives a genotype and iterates over its genes. All the genes have the same mutation probability. For a gene being mutated, we first find what type of gene it is: connection, function, parameter or program output. If it is a program output, the *Output* class is used. If it is a connection gene, the *Connection* class is used. If it is a function gene, the *Functions* class is used. Recall that when we add parameters to the CGP, we must provide an initialization function and a mutation function. If it is a parameter gene, the mutation function provided will be called.

After iterating all genes, the active nodes are found again, and the fitness is recalculated.

5.3 Example

To understand how the toolbox is used to a particular problem, is now shown an example. The example is briefly described and it is detailed in Appendix B. The idea is to apply the toolbox for the symbolic regression problem of a known and frequently used 6^{th} order polynomial:

$$y(x) = x^6 - 2(x^4) + x^2. \tag{5.1}$$

The goal is to evolve mathematical function to accomplish the desired expression. To build an an application for complementing the toolbox several steps must be made starting on import the package **cgptoolbox** into Matlab workspace. Then some components have to be provided, and they are represented in Figure 5.3.

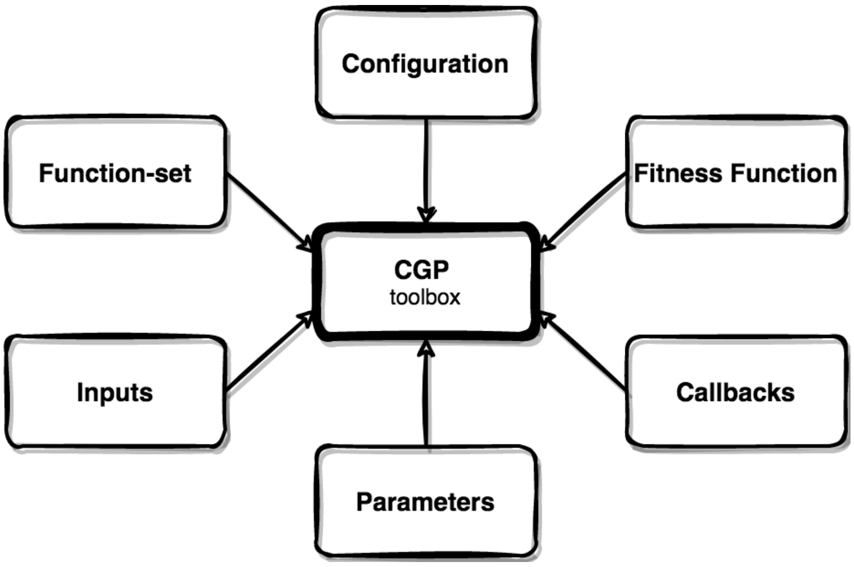


Figure 5.3: Components to provide to the CGP Toolbox.

The necessary steps for running the toolbox are the following:

- configure the CGP;
- add the program inputs;
- provide a fitness function;
- add the function-set.

Also, there are two optional steps:

- add callbacks to be executed during the evolutionary process;
- add parameters to the genotype.

The detailed process is described in Appendix B. For this particular example all the runs leaded to the desired mathematical function corresponding to the polynomial function in Equation 5.1.

5.4 Summary

This chapter described in detail the toolbox developed for implement CGP, especially built for signal processing problems like image and sound (CGP4Matlab). It was tested with a common case of a polynomial curve regression $y(x) = x^6 - 2(x^4) + x^2$, obtaining always the exact desired expression. It was also tested with periodic signals added with noise to estimate the fundamental frequency. The CGP4Matlab was presented and published in [Miragaia et al., 2018] along with a first approach to the problem of pitch estimation of piano music.

Chapter 6

Cartesian Genetic Programming applied to pitch estimation of piano sound

Multi-pitch estimation or multiple fundamental frequency estimation is the process of extracting musical notation (pitches) from a given acoustic signal. Multi-pitch estimation is a task that belongs to content-based music information retrieval. Music information retrieval (MIR) has drawn more attention due to the exponential growth of digital music [Casey et al., 2008]. Howbeit, there is a significant gap between high-level human perception and low-level signal features. Among all musical instruments, the piano is one of the most popular instruments worldwide and one of the most complex in what concerns pitch variety and number of simultaneous notes [Emiya, 2008]. These are the main reasons that motivate us to research the multi-pitch estimation of piano sounds.

A single piano note is a monophonic sound comprising a quasi-harmonic spectrum [Olson, 1967], where the lowest frequency of the harmonic series corresponds with the perceived pitch or fundamental frequency (F0). The combination of multiple piano notes (monophonic sounds) results in a polyphonic sound with multiple pitches. MPE refers to determination of the underlying pitches of an obtained polyphonic sound. Unlike mono-pitch estimation, multi-pitch estimation deals with issues such as the source number ambiguity and the octave ambiguity Figure 6.1 shows an example where the spectrum of a single piano note C4, Musical Instrument Digital Interface (MIDI) note number 60, is almost identical to the spectrum of the two-key combination of C4 and C5. Therefore, multi-pitch estimation is a challenging problem. Although there has been a lot of research devoted to it, it still remains unsolved.

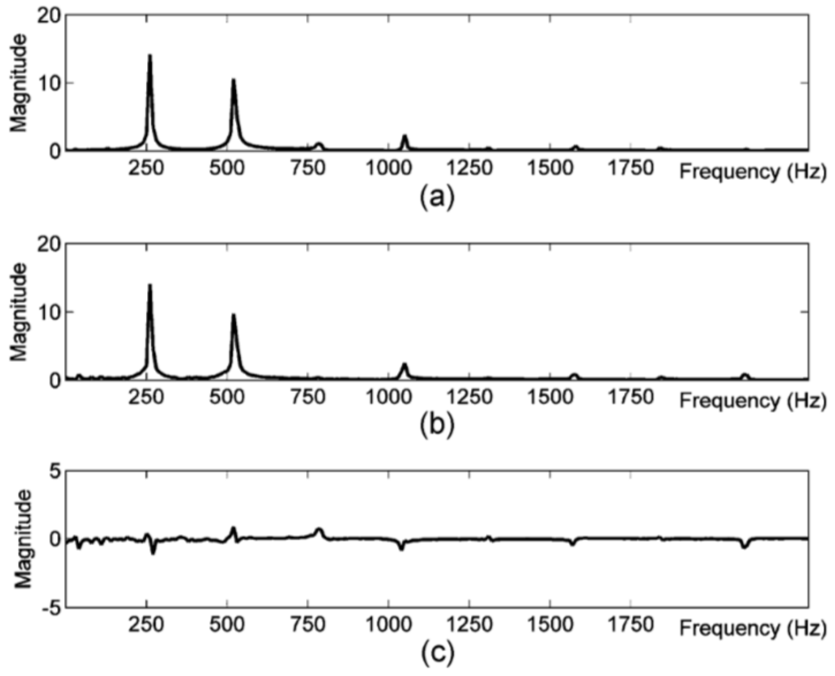


Figure 6.1: Example of source number and octave ambiguity: (**a**) Spectrum of C4. (**b**) Spectrum of the mixture of C4 and C5. (**c**) Signal difference between (**a**,**b**). Adapted from [Lee et al., 2012]

In general, multi-pitch estimation approaches and algorithms assume that there might be more than one harmonic source in the same short-time signal. As explained by [Yeh et al., 2009], that signal may also be expressed as a sum of harmonic sources plus a residual (The residual, $z[n]$, comes from components that are not explained by sinusoids, for instance, the background noise, spurious components, or nonharmonic partials.):

$$y[n] = \sum_{m=1}^{M} y_m[n] + z[n], M > 0 \text{ with } y_m[n] \approx y_m[n + N_m], \tag{6.1}$$

where M is the number of harmonic sources, n represents the discrete time, $y_m[n]$ is the quasi-periodic part of the mth source signal, N_m represents the period of the mth source, and $z[n]$ is the residual. According to Fourier series and the signal quasi periodicity, $y[n]$ can be written as follows:

$$\begin{aligned} y[n] &= \sum_{m=1}^{M} \Big\{ \sum_{h=1}^{\infty} A_{m,h} \cos(h\omega_m n + \phi_{m,h}) \Big\} + z[n] \\ &\approx \sum_{m=1}^{M} \sum_{h=1}^{H_m} A_{m,h} \cos(h\omega_m n + \phi_{m,h}) + z[n]. \end{aligned} \tag{6.2}$$

Last-step approximation is common for practical usage: a finite and small number of sinusoids H is commonly used to approximate a quasi-periodic signal.

The main difficulty associated with MPE is dealing with the modeling of $y[n]$. This task implies estimating the number of harmonic sources and their related F0s. Decomposition of the observed signal into an unknown number of model sources not only is a problem of pattern matching but also can be seen as a classification problem, i.e., identifying the most likely combination of F0s for the modeling of $y[n]$.

Evolutionary algorithms are very successful in solving both pattern matching and optimization problems [Goldberg and Holland, 1988]. However we may decompose this problem into several classification problems if we consider the process of recognition the presence of one particular note or pitch as a classification problem. Thus, we have many classification problems each one for one note or piano key in the case of piano sounds. This led us to propose a multi-pitch estimation system based on an evolutionary approach: genetic programming (GP). Among all GP variants available, we decided to innovate by addressing this problem using Cartesian genetic programming (CGP), which has already proven to be able to solve classification problems related to signal and image processing [Harding et al., 2013]. Although we may consider MPE as a set of classification problems, using CGP we may consider each one as an optimization problem (optimize a search). In fact for each pitch we will evolve one expression or set of mathematical functions that maximize one fitness function, i.e that are capable to classify correctly using a classification evaluation metric piano sounds.

6.1 Overview

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, a "Divide and Conquer" approach. Our system's classifiers are 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.

Genetic Programming is aimed at creating computer programs capable of solving a given problem. Cartesian Genetic Programming builds programs in a form of graphs. To address the MPE problem, our CGP system generates programs in a form of graphs using a set mathematical functions (function set). This way, each each individual will be an evolved graph encoding a complex mathematical expression.

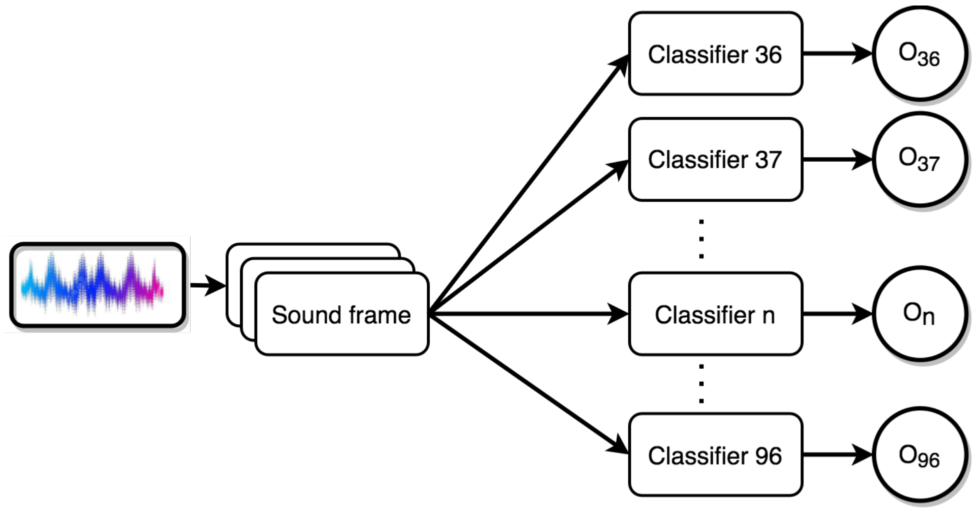


Figure 6.2: Block diagram of our MPE working system with multiple classifiers: 61 classifiers (from 36 to 96)

The proposed estimation system initially developed for piano music consists on a set of classifiers, where each piano note is identified by a CGP evolved classifier (Figure 6.2). 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 according to the particular case of a 61 key piano. This can be easily generalized for any other piano with different number of keys . Each one of these classifiers uses several inputs, all of them deriving from the acquired audio signal and return one binary output, indicating if the corresponding piano note is present or not in the given signal. Basically, a sound vector is sampled, this process can be made iteratively or using a note onset detection system. Then, the inputs are computed from the original sound vector using several signal processing techniques. These inputs are then used by the classifiers CGP functions, to accomplish an output vector. This vector suffers a binarization process to obtain a final binary output.

In the proposed CGP system, each piano note is identified by a classifier previously evolved using CGP. 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. This grid of nodes has three main properties as explained before: n_c (columns), n_r (rows) and *l-back* (how many previous columns of cells may have their outputs connected to a node in the current column). Depending on n_r, n_c and *l-back*, a wide range of graphs can be generated. When $n_r = 1$ and *l-back* $= n_c$, arbitrary directed graphs can be created with a maximum depth. Choosing these values imposes the least constraints. Therefore, this is the best and most general choice [Miller, 2013]. This choices are fundamental to start the first stage of an EA based system, where evolution takes place; the training stage.

Basically, there are two different stages: the training stage and the test or working stage. Both stages have processes in common and they share the majority of the blocks illustrated in Figure 6.3, which depicts the training process. The training stage is crucial and it is responsible for the learning process that conducts the evolution of the individuals. The evolution leads to the final classifiers, which are mathematical expressions evolved through CGP.

6.2 Training

To develop the CGP based classifier system, there is always a learning stage. In general this stage uses known examples i.e elements extracted from a data-set with labels. In this case those elements are sound signals of piano music or piano chords completely labeled with the pitches or notes presented at any instant. This set is called the training set. The learning stage also known as training stage is the first stage of all process and consists in evolving the classifiers trough a training process. The training process used is summarized using a block diagram in Figure 6.3. To perform the training stage a sound data-set is required, composed by several piano sounds properly labeled with the 'ground truth'. First the system extracts audio sized frames from audio files or file then, some preprocessing is applied on the acquired audio frames to generate the desired system inputs for all the classifiers. Those inputs, along with the function set and CGP parameters are used in the CGP block, where the classifiers are evolved independently using our CGP toolbox [Miragaia et al., 2018]; during this evolutionary process, each classifier generates an output vector. Then, this output vector undergoes a binarization process to obtain a final binary output. During the next step, a fitness function is computed to evaluate the quality of the corresponding classifier using F-measure. The engine of the implemented evolutionary process is the CGP block: it is responsible for the evolutionary process that is done during the training stage to accomplish one final classifier for each piano note: a graph of mathematical functions encoding a mathematical expression capable of detecting the presence of the sound produced by the corresponding piano key.

Besides the implementation of the CGP algorithm many other decisions had to be made and processes hat to be implemented. It was employed and implemented a set of preprocessing tasks (Section 6.3) to generate the system inputs, and had to figure out which inputs to use and how to represent them (subsection 6.4). It was also necessary to create a gene encoding process where each gene has a specific purpose (subsection 6.5), it was decide which kind of mutations to implement, (subsection 6.7) and which evolutionary strategy (section 6.8) to use as well. We also developed a binarization strategy (section 6.9) for the output and fitness computation (section 6.10).

6.3 Preprocessing

One important decision in designing classifier systems is defining the inputs and their structure. The original sound signals acquired from wave 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 in Sec-

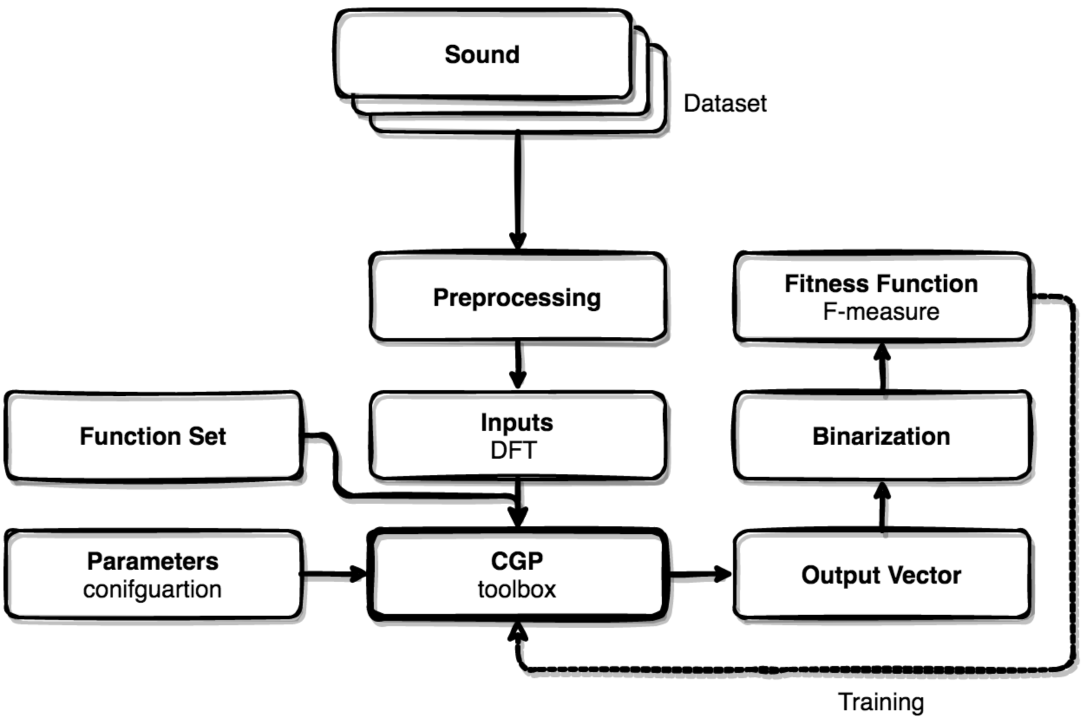


Figure 6.3: Block diagram of the CGP system with feedback for training stage: training stage starts with a signal pre-processing over the dataset, it generates the inputs, the system engine is CGP toolbox, that produces the output vector, then binarization and fitness evaluate the individuals and feedback for CGP toolbox to proceed to the next generation of the evolutionary process

tion 2.7, the polyphonic signals can also be expressed as a sum of harmonic sources plus a residual (Equation 6.1). Thus, in frequency domain we have the information we need about the Fundamental Frequencies (F0) that compose the acoustic input signal. This stresses for a need of converting the input audio signal to the frequency domain. However, some preprocessing tasks have to be made to the original sound signal to prepare it for a domain change. The first step is the sound frame or sound sample extraction. The goal is not to use an entire sound signal or music but extract small time portions of the signal with a constant size. The pitch estimation analysis will be made frame by frame. There are 2 common ways of perform AMT using MPE, one is frame-level, i.e. the sound signal is decomposed in small time frames and the MPE is performed frame by frame, the other is note-level and known as 'note tracking', uses onset detection and pitch estimation. Both approaches need to perform pitch estimation and use sized sound frames to do it.

The extraction of a sample frame is illustrated in Figure 6.4. A sound signal of a piano chord, D2+C#3+A#4, with $\approx 1s$ is used as base. In red is the part of the sound signal that will be extracted and used for MPE, it is $\approx 0.1s$ long. The time instant for the sampling process was triggered by a rudimentary onset detection process based

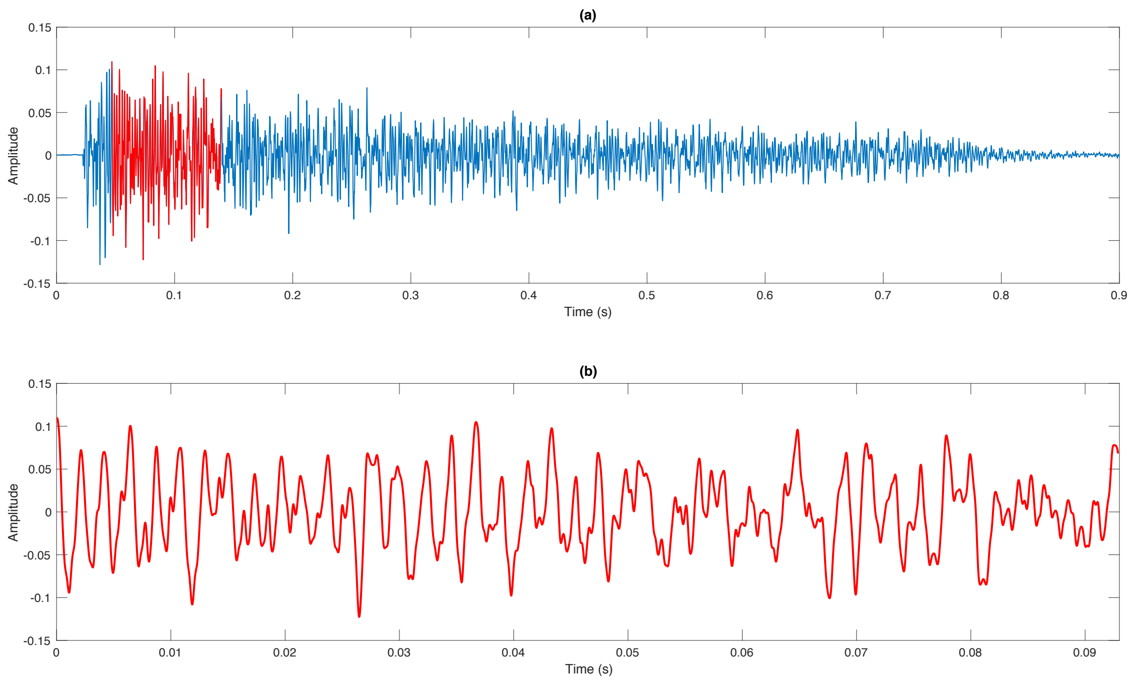


Figure 6.4: Sound frame extraction: (**a**) Piano sound signal 1 second long; (**b**) sound frame extracted of $\approx 0.01s$

on peak detection. The main concern on frame extraction from the original sound is the size of it. Because the size of the vector in the time domain is directly linked with the size of the resulting vector after a domain change, regarding that the original sound signal is in time domain and it will be converted to the frequency domain using the DFT through FFT algorithm. The relation between signal properties (time vs frequency) was addressed in Section 2.4.6. Commonly, the piano original sounds have a sampling rate $f_s = 44100Hz$, this means that in 1 second of signal we have 44100 samples, in a music this is an huge amount of time to make pitch estimation because in one second many notes may appear (onset) and disappear (ofset), thus it is important to find out a more convenient and smaller time frame to perform the signal MPE. Although choosing a very small time frame brings difficulties when analyzing the frequency domain signal, because the resulting vector in frequency maps the same frequency range with fewer elements, this constrain reduces the frequency resolution, each vector element represents a higher range of frequencies, it means a loss of precision in the frequency domain. Another relevant fact is the FFT algorithm, presented in Section 2.4.1 used to compute the DFT, it was designed to be fast and preferentially to work with arrays size of 2^n. Therefore, using time frame arrays of 4096 becomes a common choice, [Reis et al., 2012]. A time frame of 4096 represents $\approx 0,093s$ of time signal using Equation 2.40.

The second part of the preprocessing consists on windowing and applying the DFT to the time frame, the signal representation in frequency domain contains contains a lot of useful information for MPE and also gives a good basis for signal processing common

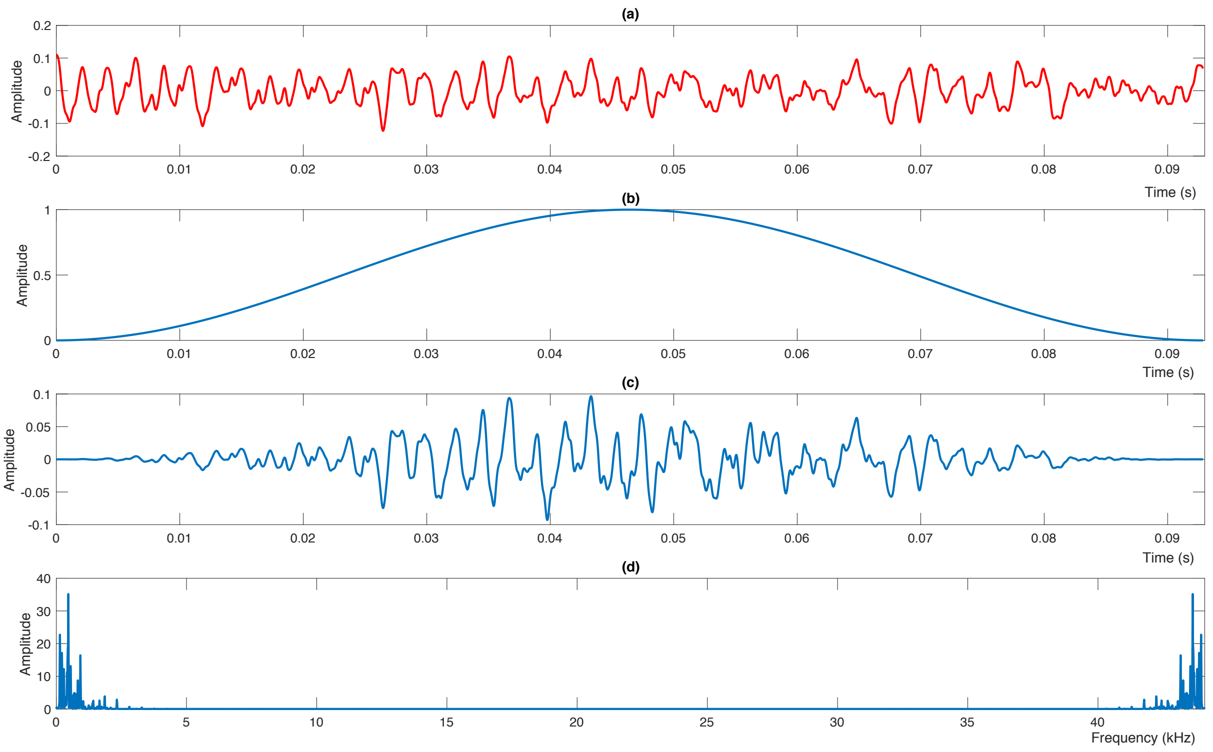


Figure 6.5: Preprocessing: (**a**) acquired audio frame in time domain; (**b**) Hanning window representation; (**c**) windowed audio frame; (**d**) windowed signal represented in the frequency domain after applying the DFT.

tasks. The transformation process is depicted in Figure 6.5 using an acoustic sound of a piano chord with pitches 38, 49 and 70. The original sound frame with 4096 samples is initially windowed, process detailed in Section 2.4.5, this process minimizes the naffest effect in the spectrum caused by the spectral leakage due to the discontinuities of truncated waveforms resulting from the frame extraction. The windowing process uses the Hanning window with 4096 points depicted in Figure 6.5 (b) plot. It is the result of the signal of the following equation:

$$w[n] = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right). \tag{6.3}$$

where $w[n]$ is the Hanning window signal and N is the number of points and it is equal to the number of samples of the signal to window, 4096. Then the original $x[n]$ signal is point wise multiplied for the window signal $w[n]$:

$$x_w[n] = w[n].x[n]. \tag{6.4}$$

resulting windowed signal $x_w[n]$ illustrated in Figure 6.5 (c). This windowed signal is transformed to the frequency domain using DFT according to the Equation 6.5:

$$X[k] = \sum_{n=0}^{N-1} x_w[n] e^{-j\left(\frac{2\pi}{N}\right)nk}, (k = 0, 1, \cdots, N-1), \tag{6.5}$$

Thus the resulting signal $X[k]$ in frequency domain has also a $N = 4096$ and it is plotted in Figure 6.5 (d). It is important to underline that the transformation using the DFT produces elements in the complex domain ($\mathbb{C}$), therefore the signal graphical representation was made using the absolute value of the complex numbers.

The signal $X[k]$ is represented in frequency and it is 4096 long, it contains information of about 0.093 seconds of sound, it has a frequency resolution:

$$\Delta f = \frac{f_s}{N} \Leftrightarrow \Delta f = \frac{f_s}{N} = \frac{44100}{4096} = 10.77Hz. \tag{6.6}$$

This means that each vector bin represents $\approx 10.77Hz$. We also know that a time signal cannot be uniquely represented for frequencies above $\frac{f_s}{2}$ (Nyquist frequency), where f_s is the sampling frequency of the sequence in this particular case 44100 Hz. Observing the plot in Figure Figure 6.5 (d) we may observe some periodicity. 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 anti symmetry 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, Figure 6.6.

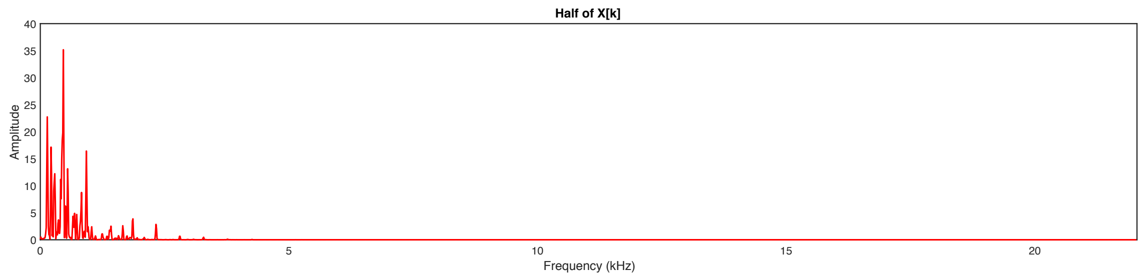


Figure 6.6: Half frequency transformed signal $X[k]$ with 2048 bins, corresponds to frequency interval $[0; \frac{f_s}{2}]$, $f_s = 44100Hz$.

6.4 System Inputs

For any EA is very important to correctly decide which are the system inputs because they carry the information that the system needs. In a Cartesian Genetic Programming system that decision should take into account the relevant information to the learning process as well as the desired output and the functions inputs of the function set because the inputs will be arguments of the functions available for usag:, the functions of the function-set. The CGP system for piano music works with sound signals, the relevant information about pitch is in the frequency domain and each sound signal is represented by a vector or array of complex numbers. This algorithm inputs are obtained from the preprocessing system. 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 93 milliseconds duration. Each complex number may be represented in the Argand plane using to different type of coordinates: Cartesian and Polar, (Section 2.4.3). Thus, the complex number vector $X[k]$ may be represented as 2 vectors using Cartesian coordinates and 2 vectors with polar coordinates:

- $\Re\{X[k]\}$ - Real part of cartesian coordinates;
- $\Im\{X[k]\}$ - Imaginary part of cartesian coordinates;
- $||X[k]||$ - Absolute value or the radius of polar representation;
- $\measuredangle X[k]$ - Angle of the polar representation.

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.

In Figure 6.7 four inputs are represented as real value vectors. In The CGP system the inputs will be the arguments of some computational nodes that also contain a function encoded that may use one or more inputs as arguments. Thus it is important to have inputs the most generalized as possible to avoid constrains to some functions. These inputs are also normalized in amplitude to a max value of 1, these brings benefits to the system and its training because it tends to learn to ignore sound amplitudes scaling. Summarizing, Figure 6.7 shows the 4 system inputs, generated from the original piano sound in Figure 6.4 containing the chord D2+C#3+A#4 (pitches 38+49+70) trough a pre-processing and an inputs generation process.

6.5 System Encoding

The process of encoding a Cartesian Genetic Programming system consists on defining the way the evolutionary structure, graph of nodes and the nodes are encoded, where each individual, a graph himself, is a possible solution. In this CGP approach to MPE each individual is a graph of encoded nodes, where each node is also encoded with integer numbers and real parameters. Cartesian Genetic Programming contains three node types: input nodes, function nodes and output nodes. To tackle this problem a system was

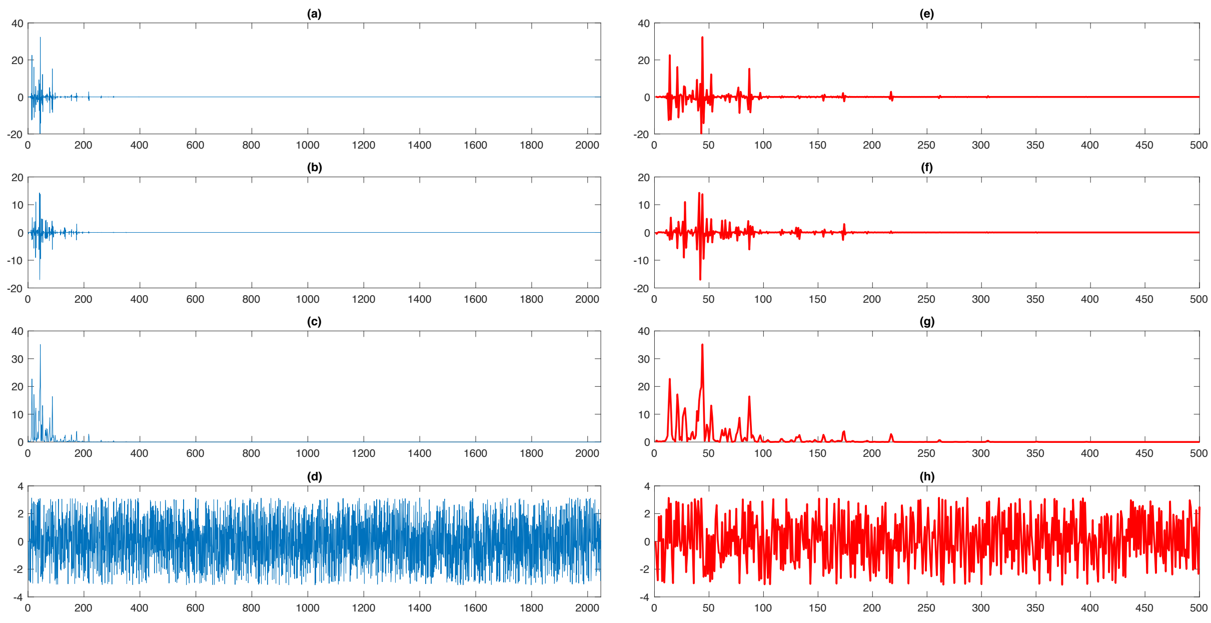


Figure 6.7: System input vectors extract from $X[k]$ frequency based signal, (**left**) entire vectors with 2048 elements (bins), (**right**) the first 500 elements of the entire vectors: (**a**,**e**) Real pat; (**b**, **f**) Imaginary pat; (**c**, **g**) absolute value of the complex number (radius); (**d**, **h**) angle of the complex values;

designed with 4 input nodes and only one output node for each classifier. The CGP graph of nodes is based on a grid of 1 row and 100 columns. Thus, each individual will have a maximum allowed of 100 computational nodes, which is a common value for this kind of approaches [Miller and Harding, 2008]. The approach used is based on the grid of nodes in Figure 6.8, in that structure there are 100 computational ($n_r = 1, n_c = 100$), and the connections are allowed until n_c levels back. To encode the 4 inputs in the graph structure 4 additional nodes were added in the first index positions, using the CGP4Matlab toolbox the output node has an optional and configurable encoding: it may be part of the output of any other computational node encoded as an index, or it is set to be the last computational node, this lats option was the mode used in the CGP system described in this chapter. The final node structure has 104 nodes, the first 4 are input nodes and the last one is where the output is extracted, so it is named as output node.

Besides the structure of the grid of nodes, it is mandatory to encode each node according to its type and purpose. Each computational node, also called function node, contains 5 different genes (Figure 6.9). There is a gene that represents the function number (F_n) from the pre-established function set represented in black and with the underline number. The next two genes encode the node inputs represented in blue also known as connection genes: they can be the system inputs or the outputs of another nodes, they are 2 regardless they are used or not by the function, the maximum arity of our function set is 2, this means that the functions of the function set have one or two arguments to perform their

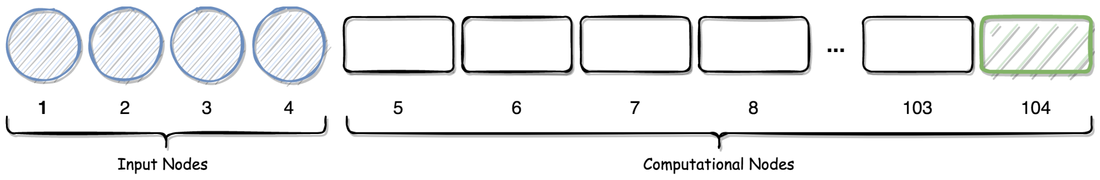


Figure 6.8: CGP grid of nodes structure: 1 line 100 columns for computational nodes: (**blue**) 4 input nodes; 100 computational nodes; (**green**) last computational node set as output node

task. Finally, there are two genes for the real parameters (P_1 and P_2), represented in red, used as parameters for some functions, that require real values besides the inputs to perform their tasks. Once again, every function node contains these kind of genes even if they do not use them. The F 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. The function set is almost comprised by filtering operations on vectors and by arithmetic operations with constants and vectors.

Figure 6.9 shows an example of 2 computational nodes with different genotypes. The node represented on the left is a computational node which uses the function number 4 and 2 real parameters (0.4 and 7.2), this note uses 2 inputs codified in the form of connection genes (3 and 1), this means that this node is connected to node index 3 and to node index 1. Observing Figure 6.8, it is possible to conclude that those 2 indexes correspond to 2 inputs of the system, because in this case the first 4 nodes are reserved to the inputs. This node is indexed as node number 6. On the other hand, the node depicted in the right side of the Figure 6.9 is the node index 10. This computational node has the function number 19 and uses as inputs the nodes indexes (6 and 9), this means that the output of node 6 is used as input in the node 10. The remaining 2 genes are the real parameters.

6.5.1 Real Parameters Genes

The CGP common implementations only use integer numbers as genes, however in some particular problems it is useful to use real numbers to encode parameters. This MPE problem that evolves a classifier composed by several chained mathematical functions, filters and direct and inverse transforms is a special case that requires real parameters for a correct work of some used functions. This real parameters should not be fixed or constant during the evolutionary process, the parameters value have direct impact in the way some functions produce work or produce the output. For instance, a low pass filter needs at least a parameter that defines the cutoff frequency. If that cutoff frequency is used in an system that evolves, it is important to allow changes or mutations in the parameter that represents that same cutoff frequency. Besides this particular function, there are many

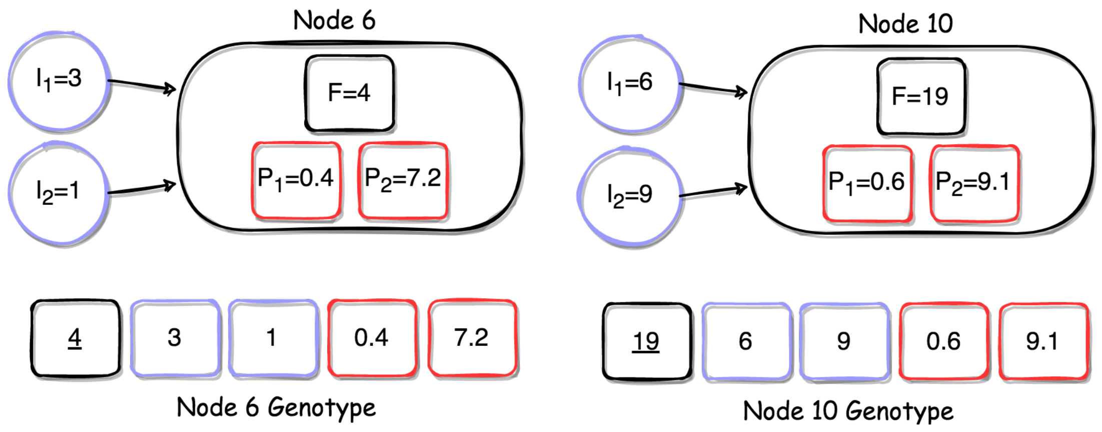


Figure 6.9: 2 Nodes genotype with genes meaning: (**left**) node index 6 uses as inputs 2 system inputs 1 and 3; (**right**) node index 10 uses as inputs 2 computational nodes 6 and 9; (**black**) function gene; (**blue**) connection/input genes; (**red**) real parameters genes

more functions in the function set that require the use of real parameters and should be allowed their mutation. As depicted in Figure 6.9 the node genotype uses 2 different real parameters for different purposes. 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 (i.e. values for arithmetic operations with vectors), 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_1 to a fix interval. Each function is responsible for mapping that fix interval to the correct domain for its own purpose. The parameter P_1 of each function with its own range is normalized into $[0;1]$:

$$P_F = P \times (b_F - a_F) + a_F,\ P \in [0;1] \tag{6.7}$$

Where P is the values of the gene P_1 for the correspondent computational node and P_F is the real value used by the function F to accomplish its task and $P_F \in [a_F; b_F]$. By using this technique, the actual value of any parameter can be seen as a number between 0 and 1 or a percentage of a defined interval and the mutation process becomes standard and easier.

6.5.2 Gene Ranges

All the genes have an interval of possible values according to their specific type. The various types of genes used in the computational nodes that codify the classifier, are categorized into 3 types:

- **Function genes** - a number to encode a specific mathematical function from a function set;
- **Connection genes** - a number to encode a connection via node input, i.e codifies the index of the node or system input that will be connected to the actual node. Are used 2 as maximum arity of the functions;
- **Real parameters genes** - A number to codify a parameter used by a specific function to perform its task, there are to different real parameters for every function, p_1 codifies a point in an interval normalized, and P_2 codifies an arithmetic constant used in arithmetic functions.

All computational nodes have in their genotype 3 types of genes. Depending on the type of gene and its purpose it can assume different range of value. The connection genes are integer values, $I_1, I_2 \in \mathbb{N}$ and they may assume values depending on the node index where they are placed and the configured l (levels back), because each node can only be connected to the nodes in previous columns of the CGP grid. In this case the levels back value is the number of the CGP grid columns (n_c), in the end of the day each connection gene can have values from 1 to its node index minus 1. The function gene of each node represents the function used in that particular node, it is the index of the function in the function look-up table. Thus the function gene (F) may assume values between 0 and n_F, the number of functions in the function set, it is also is an integer value. Finally, the real parameters P_1 and P_2 have distinct purposes and distinct ranges. $P_1 \in [0; 1]$ and $P_2 \in [-c; c]$, where c in this particular approach takes the value 15. The value ranges for each type of genes are fundamental to understand their purposes as well as to define how they mutate during the evolutionary process.

6.6 Function Set

The function set is the set of functions that the CGP system has available to use in each computational node. The main idea is to build a look-up table where each line corresponds to a function and where every function is indexed with an integer number. As mentioned before, each computational node has a gene that indicates which function of the function set should be used on that node. Table 6.1 shows the function set with a look-up table that was used in the first approach to the problem. The maximum function arity is 2, this way every functions are prepared to receive as arguments one or two vectors and all of them return one vector as output. Thus, there are no constrains problems with the inputs domain or chaining functions.

Table 6.1: Function set - lookup table.

Index	Function	Description
1	SPAbs	Absolute value
2	SPBPGaussFilter	Band pass Gaussian filter
3	SPConvolution	Convolution
4	SPCos	Cosine Function
5	SPDivide	poit 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 Function
20	SPSubtract	Subtraction
21	SPSum	Sum
22	SPSumConst	Sum with a constant
23	SPThreshold	tresholding
24	SPTimes	Multiplication

The function set is basically composed by arithmetic functions with constants or vectors, transform related functions and filtering operations. Our function set is basically composed of filtering operations on vectors and arithmetic operations with constants and vectors.

SPAbs This function returns the absolute value of each data point of the first vector received as argument.

SPBPGaussFilter performs a band pass filter of the first input, centered in the frequency represented by the first parameter.

SPConvolution performs a convolution between the first input vector and the second input vector.

SPCos applies the cosine function to each data point in the first vector and returns it.

SPDivide divides each point in the first input vector by each data point in the second input vector.

SPFFT applies the DFT to the first input vector, and returns its absolute value.

SPGaussfilter applies a Gaussian filter to the first input vector. Sigma (first parameter) can evolve between 1 and 10, depending on the interval parameter encoded in the node.

SPHighPassFilter applies a high pass Gaussian filter to the first vector input, using first real parameter to represent the cutoff frequency as a point of the vector.

SPIFFT applies the Inverse DFT to the first input vector, and returns its absolute value.

SPLog returns the natural logarithm of the first input vector.

SPLog10 returns the absolute value of the common logarithm applied to the first input vector.

SPLowPassFilter it filters the input vector with a low pass Gaussian filter using a cutoff frequency based on the first real parameter.

SPMedFilter applies a median filter to the first input vector. The percentage of signal that will be filtered is in a range between 3% to 10%, depending on the first parameter.

SPMod returns the remainder after division of the first input vector by the second input vector.

SPMulConst return the multiplication of the first vector by a constant (second parameter).

SPNormalizeMax returns the normalization of the first input vector by the maximum value in the vector.

SPNormalizeSum returns the normalization of the first input vector by sum.

SPPeaks returns the first input vector with all non-peak values set to 0.

SPSine applies the sine function to each data point in the first vector and returns it.

SPSubtract returns the subtraction of the first input vector by the second input vector.

SPSum returns the sum of the first input vector with the second input vector.

SPSumConst returns the sum of the first input vector with a constant (second parameter).

SPThreshold sets all values of the first vector below a certain threshold (first parameter) to zero, and returns its value.

SPTimes multiplies the first vector by the second vector.

The computational node genotype is composed by genes, however depending on the function gene the computational node may have from 2 to 5 genes with actual meaning in the phenotype, thus there are genes that may be ignored. There are functions that use 2 arguments (inputs) and others uses only one, being the remain ignored for phenotype purposes. There are some functions from the function set that use one real parameter others use two and there are some that do not use real parameters. Although, even the ignored genes (not used in phenotype) are kept in the genotype and mutated if selected, because if the function gene changes they have the chance to be used by another chosen function, in the same computational node.

6.7 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, thus the genetic variety relies on the mutation operator. The mutation process depends on two different stages, ruled by different probability distribution 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 to 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 (if the option *'last node'* is not set). 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 $(P1, P2)$. The mutation of the real genes (function parameters) is done using the normal distribution in order to address the entire range. The Mutation process of one particular gene (gn) with a configurable mutation probability (p) is systematized in the pseudo-code of the Algorithm 5.

The idea is to use an uniform distribution in the interval $[0; 1]$ to generate a random value (p') for each gene (gn) this way, if the generated random value is lower than (p) the gene is set to be mutated. The mutation process follows different probability distributions and different acceptable ranges, according to gene type. The system mutation peculiarity is the mutation process of the real parameters. As mentioned the real parameters uses the Normal distribution. a normal (or Gaussian or Gauss or Laplace–Gauss) distribution is a type of continuous probability distribution for a real-valued random variable. The general form of its probability density function is:

$$f(x) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}},\ x \in \mathbb{R} \tag{6.8}$$

Algorithm 5 Mutate gene (gn) with probability p

```
$p' \leftarrow$ random number in $[0;1]$
if $p' < p$ then
    $gn \leftarrow$ gene set to mutate;
    if $gn$ is connection gene then
        mutate $gn$ randomly in (node indexes + inputs)
    else if $gn$ is function gene then
        mutate $gn$ in function lookup table
    else
        mutate $gn$ using normal distribution          ▷ distinct for $P_1$ and $P_2$
    end if
end if
```

where $f(x)$ represents the probability density function of x variable that represents the parameter values, with a normal distribution. This function is also represented as $N(\mu, \sigma)$, where μ is the mean and σ is the standard deviation. The system contains to configurable parameters σ_1 and σ_2, these parameters are used to set the standard deviation of the function $f(x)$ for the real parameter P_1 and P_2. Note that they have different range of values as detailed in Section 6.5.2. Lets consider the mutation process of parameter P_1, since the process for P_2 is similar. The initial value for P_1 is obtained using the random value from normal distribution $N(\mu, \sigma)$, where $\mu = 0.5$ half of the interval $[0;1]$, and $\sigma = \sigma_1$ the configurable parameter (usually used as 0.3). To follow a simple example, (Figure 6.10) we may consider an actual value for $P_1 = 0.5$, if that gene is set to be mutated the blue line will be the probability density function, $N(\mu = 0.5, \sigma = 0.3)$ to generate a mutate P_1'. The new random generated value, $P_1 = 0.71$. If that value will be choosen to mutate again, the new value will be generated according the red curve in Figure 6.10, that is a probability density function $N(\mu = 0.71, \sigma = 0.3)$, i.e, a normal function centered in the actual values of the parameter and with the same standard deviation value.

It is also important to ensure, during the mutation process, that the P_1 is always in the range of [0;1]. The use of a normal distribution centered in the actual value of the parameter for generating a random new value for it, may lead to a values out of range since in Equation 6.8 $x \in [-\infty; \infty]$. Algorithm 6 describes the mutation process of the real parameter P_1, using the normal probability distribution centered in the actual value of the parameter.

The Algorithm 6 needs to get the actual value of the parameter gene (P_1) and the standard deviation value parameter stored in the configuration file (σ_1), to use these 2 parameters to manage the random process of generating a new value for P_1, temporarily stored in P_1'. The random new value is generated using the Gaussian probability density function with mean $\mu = P_1$, this means that the normal function is centered in P_1 and uses a standard deviation value constant, extracted from configuration $\sigma = \sigma_1$. Finally, if the generated random value is out of acceptable range, [0;1] the process is repeated until a valid new value is reached. This mutation process is used for the other real parameter

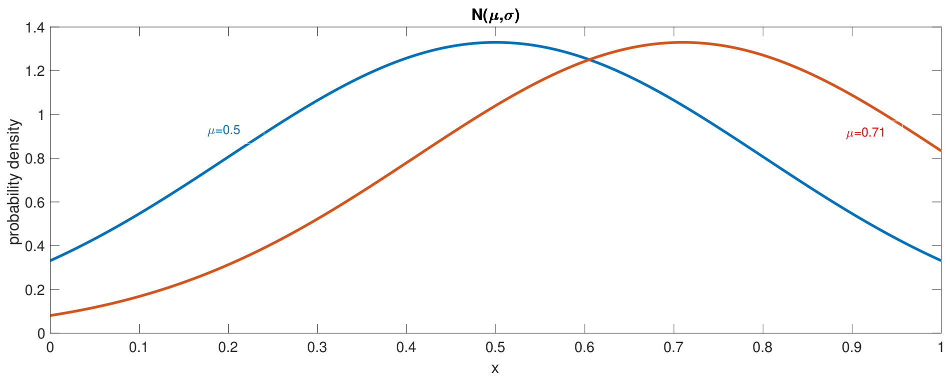


Figure 6.10: probability density functions of normal distribution: 2 consecutive normal distribution functions for mutation, (**blue**) centered in $\mu = 0.5$ in red centered in $\mu = 0.71$, both with the same standard deviation 0.3

Algorithm 6 Mutate real gene P_1 using normal distribution

```
1: get actual $gn$ value $P_1$
2: get from configuration file $\sigma_1$
3: $\mu \leftarrow P_1$
4: $\sigma \leftarrow \sigma_1$
5: do
6:     generate random value $P_1'$ using $N(\mu, \sigma)$
7:     $P_1 \leftarrow P_1'$
8: while $P_1 > 1$ OR $P_1 < 0$
```

P_2 and for other system parameter, the threshold. The parameter P_2 is mutated exactly as is shown in Algorithm 6, with different values for configurable standard deviation (σ_2) and with a range of valid values also distinct, instead [0;1] it is used the interval [-15;15].

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.

6.8 Evolutionary Strategy

The Evolutionary Strategies have been presented in in Chapter 4, and the CGP usual EA was detailed in Section 5.1.2. Thus the EA used in this CGP system for MPE is based on the simple evolutionary algorithm known as $1 + \lambda$ [Eigen, 1973]: the new offspring is obtained promoting the fittest individual and generating λ new individuals trough mutation. Therefore, mutation has a crucial role in this system, it is the only genetic operator used to guarantee evolution and it is detailed apart in previous Algorithms 5 and 6. Another particularity of the ES used, is the fact that 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. According to [Goldman and Punch, 2015], an empirical value for

λ is 4, which was the value configured in our configuration file for extensive use in our system. Being an evolutionary strategy a non-deterministic process and with a lot of aleatory components it makes entire sense to perform multiple evolutions of a classifier, i.e. make multiple runs of the evolutionary process. All this adaptations and details are documented in Algorithm 7, encoded with multiple runs.

Algorithm 7 Algorithm ($(1 + \lambda)\, EA$) encoded with multiple runs

```
1: $r \leftarrow 0$;
2: while $r < mr$ do;
3:     $g \leftarrow 0$;
4:     Set current individual $I_0$ as the best of $\lambda$ individuals created randomly;
5:     $F_g \leftarrow$ fitness of current individual $I_0$
6:     while $g < mg$ and $F_g < f$ do
7:         for i = 1 to $\lambda$ do
8:             Create a copy $x_i$ of current individual $I_g$;
9:             Mutate each gene of $x_i$ with probability $p$;        ▷ mutation algorithm 5
10:        end for
11:        Set $I'_g$ as the best of $\{x_1, \ldots, x_\lambda\}$;
12:        if $I'_g$ is better or equal than $I_g$ then
13:            $I_{g+1} \leftarrow I'_g$
14:        else
15:            $I_{g+1} \leftarrow I_g$
16:        end if
17:        $F_{g+1} \leftarrow$ fitness of current individual $I_{g+1}$ ;
18:        $g \leftarrow g + 1$;
19:    end while
20:    Save the best individual of run $r$ ($I_g$) as $B_r$;
21:    $r \leftarrow r + 1$;
22: end while
```

In Algorithm 7 there will be mr runs, the best individual of each run r will be saved as B_r. Each run consists on an evolutionary process that will continue until generation g reaches the maximum number of generation allowed, mg or the fitness of the best individual of the current generations F_g reaches the maximum fitness value f equals to 1, in this maximization case. During each run, an offspring of 4 elements x_i will be generated as a copy of the current fittest individual I_g. Each individual x_i will suffer a mutation process, using Algorithm 5 invoked in line 9. The best individual, (fittest) of all x_i is temporarily stored as I'_g. If the best individual of the offspring is better or equal than the actual fittest I_g, then it will take the place of I_g as the fittest I_{g+1}, and if the process continues it will pass for the next generation. This process ensures genetic diversity through generations, it also promotes silent mutation and neutral search.

During the evolutionary process, there is a reasonable percentage of inactive genes. Such inactive genes have a neutral effect on the genotype fitness [Miller and Smith, 2006]. However, [Vassilev and Miller, 2000] investigated in detail the influence of neutrality in CGP and shown it to be extremely beneficial to the efficiency of the evolutionary process.

For better computing performance, we also took into 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.

6.9 System Output and Binarization

The MPE system developed based on CGP is a multi-classifier system, Figure 6.2, where each evolved classifier detects the presence of each piano note (key). It follows the block diagram depicted in Figure 6.3 where the final block is the *'Fitness Function'*. The fitness function uses F-measure which requires binary output values. The output vector block, only extracts the output of the system grid of nodes that is the last node of the row, when using the *Last node* configuration for the CGP instead of random output node. At this point the system output is a vector sized equals to the inputs, in this case 2048, that needs to be transformed into a Boolean value where 1 means the presence of the correspondent note and 0 otherwise. The *Binarization* block is responsible for the process of transforming an output vector into a binary Boolean output.

The fitness function is compute for each individual (possible solution), each individual represents one specific classifier for a specific note. To use the fitness function to calculate the fitness for one classifier, we need the classification results of that classifier applied to a set of sound frames, called training set. The classification results must be '1s′ and '0s', true or false for each input sound signal for a classifier. The fitness function will use those binary results and the ground truth to compute f-measure.

The CGP system is comprised of mathematical functions whose arguments are vectors and that also return vectors. Thus, at the end of each CGP graph (output node), we have a vector of float values. The binarization process added at the end of each classifier to transform its output vector into a binary output was done using a base signal and a vector inner product. The system, depending on the classifier builds a base signal, the base signal depends on the pitch of the note that is being analysed, i.e. each classifier is evolved for one piano pitch or key number with a fundamental frequency (F_0). To calculate the F_0 for a piano key we use the following equation that gives the fundamental frequency $F_0(n)$ of the n^{th} key:

$$F_0(n) = 440 \times 2^{\frac{n-49}{12}} Hz, \tag{6.9}$$

using Equation 6.9 the fundamental frequency is mapped according the piano key number, thus the base vector is built placing a triangle in a vector size of 2048 mapping the frequency used spectrum where each bin corresponds to $10.7Hz$. The amplitude and width of the triangle is configurable and centered in the $F_0(n)$ mapped on the vector. Figure 6.11-(a) shows the base vector computed for key number 66. In order to accomplish a binary output, we use a discrete interception process between the CGP output vector normalized in amplitude $O_{cgp_n}[k]$, (Figure 6.11-(b)) and the base signal with the frequency corresponding to the pitch of the estimator, $B_{F0_n}[k]$. The first step is the normalization of the output vector in amplitude. Due to the normalization process using the maximum value of the vector it is ensured that every vector element falls into the interval [0;1], this is important for managing the threshold range values. Then, we generate the following scalar value, computing the inner product between the 2 vectors:

$$A = \sum_{k=1}^{N} O_{cgp_n}[k] \times B_{F0_n}[k], \tag{6.10}$$

where A, besides resulting from the inner product measures the discrete intersection between the two discrete signals. Making an approximation for the 2 signals to a continuous domain, the A value can be seen as the intersected area between the two signals, the entire process is depicted in the sequence of 3 graphs in Figure 6.11.

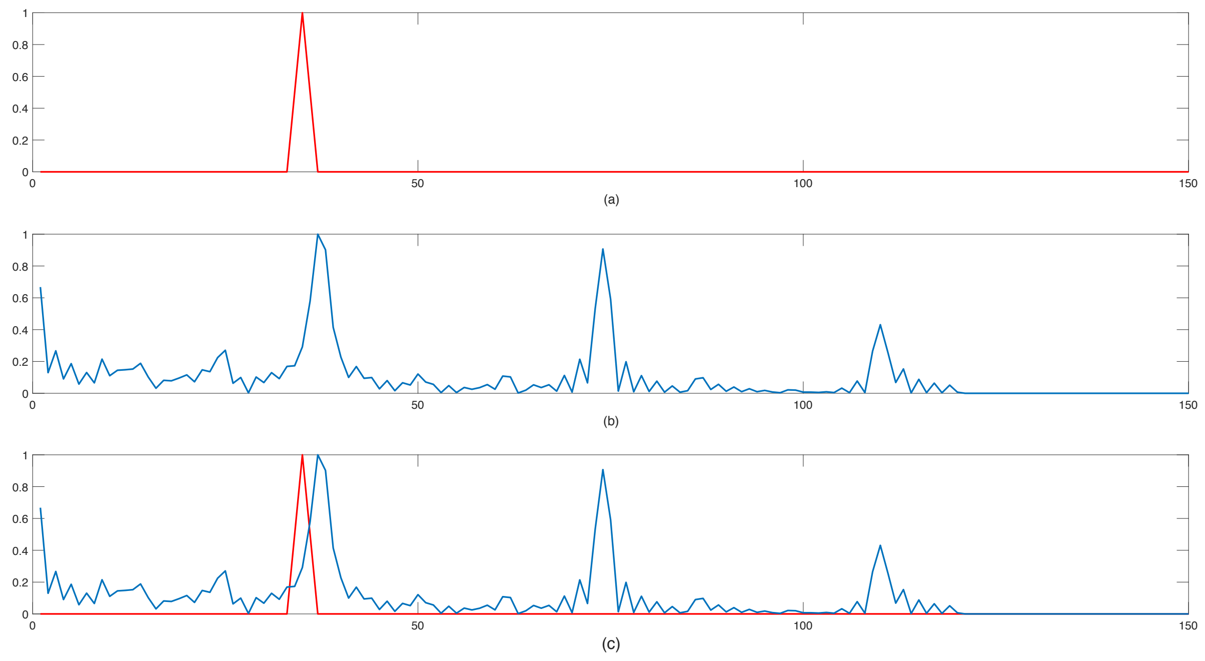


Figure 6.11: Binarization: (**a**) Base triangular vector for pitch 66; (**b**) Output vector; (**c**) Interception

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

$$T(A) = \begin{cases} 1, & \text{if } A > \theta \\ 0, & \text{if } A <= \theta \end{cases} \tag{6.11}$$

where θ is the threshold value and $T(A)$ is the Boolean system output for a classifier of a given sound sample. Remembering that 1 means a positive case and 0 a negative one.

Since both signals are normalized in amplitude, the max value for A is:

$$A_{\max} = \sum_{k=1}^{N} B_{F0_n}[k]. \tag{6.12}$$

Computing Equation 6.12, the system is able to determine the interval of values for placing the threshold for the correct system work. However, after a few experiments it becomes clear that the threshold value should be a system parameter and subject of exhaustive tuning. Since our MPE proposal is based on Evolutionary Algorithms, it was consistent to use genetic operators to reach the best value for the threshold for the problem. Therefore, the threshold was encoded as a system gene able to mutate during the training stage and consequently evolve along with all the others CGP genes. This way, besides the node genes, the threshold may also adapt to reach a better system fitness. The threshold mutation probability is independently configurable, and was empirically set to the same mutation probability of the other genes. When the system decides to mutate the threshold, its mutation process is ruled by the same algorithm of the real parameters using the normal probability distribution, Algorithm 6. The initial threshold value is set as a configurable parameter and it mutates using the Gaussian probability density function $N(\mu, \sigma)$ where σ is also a configurable parameter. The interval for valid mutations is $[0; A_{\max}]$.

6.10 Fitness

The last system block of the chain in Figure 6.3 is the **Fitness Function**. This block is fundamental in the training stage but is also used during the test stage.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. Therefore, the fitness function evaluates the quality of the evolved classifier not only during the training process but also during the test stage. In statistical analysis of binary classification, F-measure is a measure of a test's accuracy widely used since its proposal [Chinchor, 1992]

This way, during the evolutionary training process, each classifier can be evaluated according its classification. This evaluation is done using F-measure (Equation 6.13).

$$F_{measure} = 2 \times \frac{recall \times precision}{recall + precision}, \tag{6.13}$$

where precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances and defined as follows:

$$precision = \frac{tp}{tp + fp} \tag{6.14}$$

and recall (also known as sensitivity) is the fraction of relevant instances that were retrieved and define:.

$$recall = \frac{tp}{tp + fn}. \tag{6.15}$$

.

The main goal of the training or evolutionary process is maximize the F_m in order to reach the most accurate as possible classifier. The training stage stops if F_m reaches 1 or the max number of generations mg is reached. During the test phase the fitness function is also computed to evaluate the results of the evolved classifiers for a test data set. Denote that fitness function values belong to the interval $[0; 1]$.

6.11 Configuration Parameters

The configuration parameters are divided into 2 different sets, the CGP related parameters set, represented in the block **Parameters** (see Figure 6.3), which is directly connected to the CGP block, and a second set which includes all the other types of parameters, particularly those used for signal processing problems. This last set interacts directly with many system blocks e.g. **Preprocessing**, **Inputs**, **Binarization**, and also defines parameters used in the genes mutation functions.

Listing 6.1: CGP parameter values in file config.txt

```
% file conif.txt
% create a configuration struct

[cgp]
rows=1
columns=100
levels_back=100
output_type=last
outputs=1
generations=5000
runs=30
offspring=4
mutation=0.05
fitness_solution=1
fitness_operator=>=

% initialize a CGP instance with a custom configuration
cgp = cgptoolbox.CGP(configuration);
```

During this first approach to MPE using CGP detailed in this chapter, the system used the parameters in Listing 6.1. The meaning of the parameters was described in Table 5.1. This parameters are used as a configurable structure, it will perform 20 runs with 1000 generations each, maximizing the fitness function with a maximum values of 1. The grid of nodes will have 1 row (n_r), 100 columns (n_c) and 100 levels back are allowed (l) setting the last node as output node. The evolution process will use $1 + 4$ ES, ($\lambda = 4$) with a mutation probability 5% (p).

Listing 6.2: Signal parameters values in file config.txt

```
% file conif.txt
% signal configuration
[signal]
sampling_frequency=44100
samples=4096
threshold=0.5
type=record
triangular_signal=0 0.5 1 0.5 0
function_set=transcription/function-set
fitness_function=fmeasure
interval_standard_deviation=0.3
constant_standard_deviation=3
fft_samples=half
threshold_increment=0.01
polyphony=chords
inputs=real,imag,abs,angle
```

The second set of parameters used are detailed in Listing 6.2. And they are specific parameters for the MPE problem. They are used in many parts of the system and their meaning is:

sampling frequency represents the sampling frequency in samples per second used in the signal processing functions (f_s);

samples number of samples used in an extracted frame of sound vector in time domain;

threshold value used for the initial threshold when it mutates, or for a fixed threshold (without mutation);

record type of sound signal;

triangular signal when using the triangular base signal it describes the triangle width and amplitude;

function set sets the path and folder where the functions of the function set are defined;

fitness function sets the type of fitness function used;

interval standard deviation it is the value of the standard deviation (σ_1) used in the normal distribution for the mutation of the real parameter P_1;

constant standard deviation it is the value of the standard deviation (σ_2) used in the normal distribution for the mutation of the real parameter P_2; usual nominated as constant;

fft samples number of samples used in the frequency domain after applying the DFT, in this case *'half'* means 2048 and it corresponds to N, size of inputs;

threshold increment threshold increment used for incremental mutation, if used;

polyphony parameter used for defining 2 different approaches monophonic and polyphonic;

inputs this parameter is used to compute the inputs, it is a set of strings where each string is the function used to compute a specific input, this case 4 inputs.

6.12 Experiments and Results

To evaluate the quality of the system, after the configuration process, it is required a training stage for all the classifiers. After that, the system can be tested with a test dataset, the F_m is calculated and the results are analysed. In the first approach to the MPE problem using Cartesian Genetic Programming, the main concern was on investigate if the used system and technique where feasible, and how every blocks and steps worked, rather than the objective values of the final F-measure. Therefore this first set of experiments is seen as a staring point for the entire research.

6.12.1 The Dataset

To perform an evaluation of the classification system, the experiments require a dataset large enough, labeled with ground truth and used by other approaches to the same problem. The choice fell on MAPS[1] database [Emiya et al., 2010] because of all said and it was available and was free for scientific use.

MAPS provides recordings with CD quality (16-bit, 44-kHz sampled stereo audio) and the related aligned MIDI files as ground truth. The overall size of the database is about 40GB, i.e. about 65 hours of audio recordings. The database is available under a Creative Commons license. A large amount of sounds and a reliable ground truth are provided thanks to some automatic generation processes, consisting in the audio synthesis from MIDI files. The use of a Disklavier (MIDIfied piano) and of high quality synthesis software based on libraries of samples permitted a satisfying tradeoff between the quality of the sounds and the time consumption needed to produce such a quantity of annotated sounds. In order to favor generalization to many audio scenes, several grand pianos and upright pianos have been played in various recording conditions, including various rooms and close/ambient takes. It also specifies the origin of the recording, which may be high quality synthesis software based on sample libraries or a Disklavier. produced and can be stored in one 4.7GB DVD. The contents of MAPS is divided in four sets:

[1]MAPS stands for MIDI Aligned Piano Sounds and it is available for use under previous request.

- ISOL - isolated notes and monophonic excerpts;
- RAND - chords with random pitch notes;
- UCHO - usual chords from Western music;
- MUS - pieces of piano music.

The MAPS was the sound database, and from that sound it was important to built several training datasets. Each classifier needs a training set with positive and negative sounds, i.e. a set of sounds where a given note is present and a set where the note is not present. After trained the classifiers have to be tested and for that purpose were built testing sets disjointed from the training sets. The idea was to train the system and then test it with unseen data, randomly chosen and mainly, using the sub-set "RAND". This is a set were there are no musical rules or harmonic rules for chords construction. This way the system was being prepared in a more general way.

6.12.2 Training

The preliminary training tests were conducted under the configuration file presented in Listings 6.1 and 6.2. The training set was built for each of the classifiers using the MAPS data-set. Each training set includes monophonic and polyphonic sounds extracted from the sub-set ISOL and RAND respectively.

Table 6.2 shows the values of the configurable parameters summarised. The evolutionary process consisted of 30 runs with 5000 generations each, using 50 positive and 50 negative cases. The number of computational nodes is 100. The classifiers were evaluated using the F-measure (Equation 6.13).

Table 6.2: List of parameters used in the experiments.

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
Offspring	4
Mutation Probability	5%
Runs	30
Generations	5000

In Figure 6.12 is presented a bar plot for 30 runs of the training stage of classifier 60, each bar individual results of each run (i.e. evolved pitch estimator). Each bar is the sum of two bars the mean value obtained for all runs (light gray) plus the difference for the actual F-measure obtained, in dark gray. The idea is to show that all runs have similar results, the overall meaning value is also depicted with an horizontal line of about 0.945, and the calculated standard deviation is $\sigma = 0.01$.

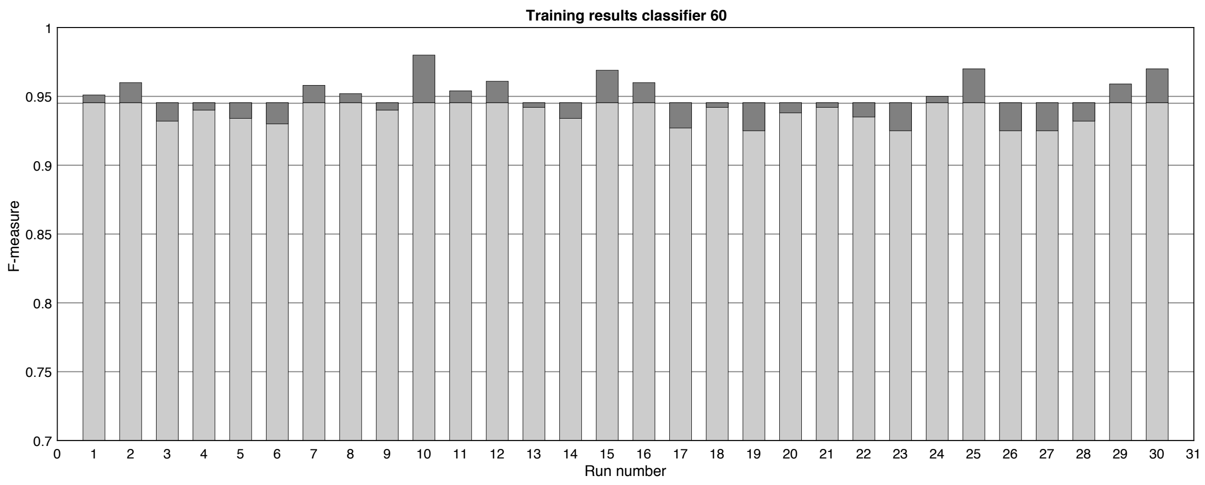


Figure 6.12: Training results obtained during 30 runs for pitch 60. Fitness values were calculated using F-measure.

The use of Cartesian Genetic Programming gives the possibility of observing in detail the mathematical result of the evolutionary process in terms of a graph of function, the phenotype. This way if the resulting genotype is decoded it is possible to see which mathematical functions are being used and how they are chained to each other as well as the inputs usage.

Listing 6.3: Evolved classifier code for pitch 60

```
% Classifier 60
%
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 )
```

In Listing 6.3 is depicted the decoded genotype program of one of the runs of the evolved classifier 60. It shows the functions used in each node as well as the inputs of each node. The first 4 nodes are the system inputs and the last node, the node 104 correspondent to computational node 100 is the system output. For instance: node 101 contains the Gaussian filter function and the function arguments (inputs) are the outputs of the nodes 36 and 87, the output of this node is the input of the last node, 104, that performs the inverse FFT. The resulting program is a set of mathematical functions, with specific parameters, over vectors - our phenotype.

6.12.3 Testing

After the training stage the system classifiers were tested with a different set, the test-set. After the training process, each classifier was tested with a different test set. Each test set consisted in 144 negative cases, (chords and single notes) and 5 positive cases, comprising a total of 149 piano sound samples.

The F-measure results for 61 classifiers, that correspond to a piano with 61 keys, are depicted in the bar plot of Figure 6.13, the key value is the F-measure mean ≈ 0.66, detailed results used to built the graph are expressed in Appendix C. Denote that this results correspond to the first approach to the problem of MPE using a multi classifier system. Both sets, training and test are relatively small because of the time consumption for evolve 30 runs for all classifiers and the main goal was to find out if the technique was feasible for the type of problem, which proved to be. However this first approach shows some problems and difficulties, it is possible to observe that the classifiers of pitches with lower fundamental frequencies have poorer results comparing with those with higher F0s. This is due to the frequency resolution of the signal representation in the frequency domain, each vector bean represents a range of frequencies of $10.7Hz$. Lower pitches have F0s more closer from each other. For instance: pitches 36, 37 and pitch 38 have fundamental frequencies $\approx 65, 69$ and $73Hz$, which means they are separated only by $8Hz$. The system needs some improvements as well as larger training and test sets, the next

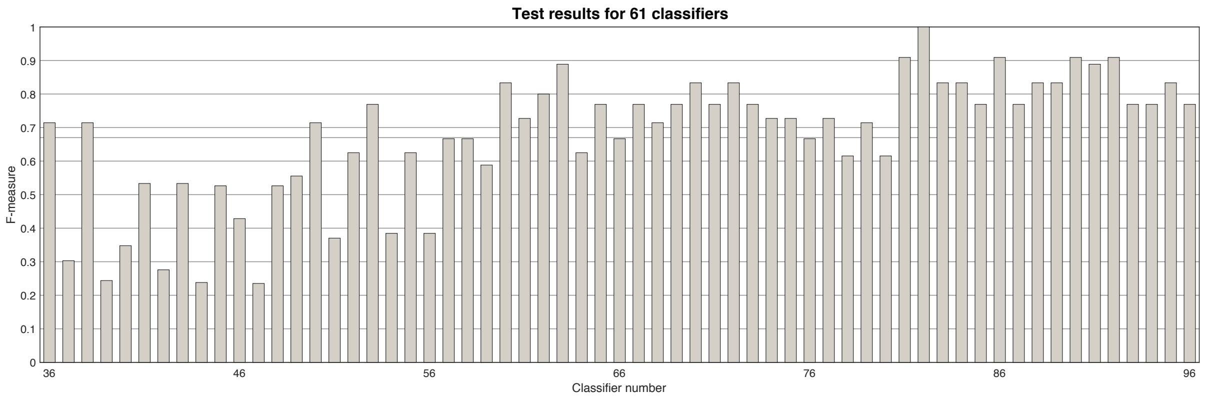


Figure 6.13: Testing results obtained for 61 classifiers from pitch 36 to 96. Fitness values were calculated using F-measure. Mean value 0.66

task is to improve the system accuracy applying some adaptations and adding some new features. This system version was named as **CGP-1** for further analysis and comparison.

6.13 Summary

In this chapter it was described the methodology used to address the problem of MPE with real audio piano signals. The system architecture was explained as well as the training process and the test process for the first approach to the problem named **CGP-1**. All the main decisions related with the implementation were addresses, from the system inputs until the fitness function and the system parameters used. Finally the first experiments and preliminary results are presented.

Chapter 7

Improving with Harmonic Mask and Data Augmentation

The system explained in the previous chapter was our first approach to the MPE of piano sounds problem, the presented results were encouraging to continue and improve the research. This chapter describes the implementation of the new features and some changes to the system that were made in order to improve the accuracy results. The system main idea remains the same, to evolve a multi-classifier system with an independent classifier for each piano note to perform multi-pitch estimation of polyphonic piano sounds. Some important changes were made, the *Harmonic Mask* (HM) and the *Data Augmentation* process stands out in terms of objective results. The new block diagram of the system is depicted in Figure 7.1. It presents two additional blocks in red compared with the diagram in Figure 6.3 and 3 other blocks highlighted in blue suffered relevant changes. Following the block chain there is a new block named **Onset Detection**: it detects when a new note appears or starts. This point is used in the next new block, **Data Augmentation** to enlarge the data-set for training purposes. After preprocessing, the inputs are extracted from the sound samples, besides the 4 inputs based on the DFT a new one is added using cepstrum: block Inputs. The block Function Set was extended with more functions to have more possibilities. Finally the block Binarization was prepared to use an harmonic mask.

During the rest of this chapter, all the improvements made in the system architecture and some implementation details will be presented. Section 7.1 explains the introduction of a new system input, section 7.2 introduces the onset detection algorithm implemented, used in this system also to perform a data augmentation process described in Section 7.3. Section 7.4 explains the extension of the initial function with the introduction of new functions, during the Section 7.5 it is shown how a new harmonic mask is used and configured. In Section 7.6 all the new blocks and changes of the system are schematized and a brief overview is made, and finally Section 7.7 describes all the experiments made and presents the results obtained.

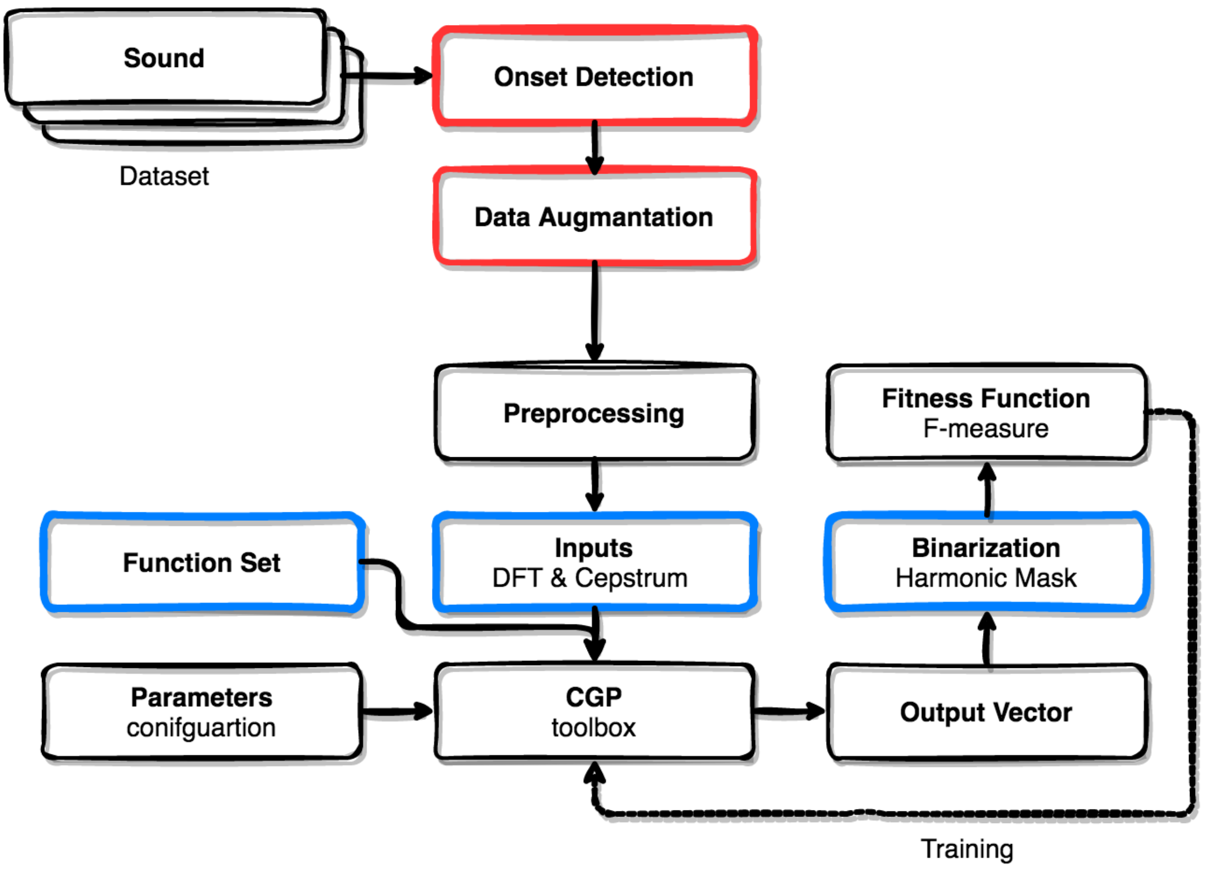


Figure 7.1: Block diagram of the improved system: (**red**) the new added blocks, (**blue**) the improved blocks with some changes

7.1 Using Cepstrum as System Input

The Cepstrum was introcuced in Section 2.6.1 as a mathematical tool for periodicity analysis in signal processing The introduction of a new input was decided when cesptrum analysis was studied.

The cepstrum is a mathematical transformation deeply used in audio signal processing in particular for pitch detection. Cepstral signal analysis is one out of several methods that enables us to find whether a signal contains periodic elements. The method can also be used to determine the pitch of a signal [Noll and Schroeder, 1964]. The cepstrum is defined as the inverse DFT (IDFT) of the log magnitude of the DFT of a signal:

$$c[n] = F^{-1}\{log|F\{x[n]\}|\}. \tag{7.1}$$

where F represents the DFT and F^{-1} the IDFT. For a windowed frame audio signal

$x[n]$, cepstrum is:

$$c[n] = \sum_{n=0}^{N-1} log(|\sum_{n=0}^{N-1} x[n]e^{-j(\frac{2\pi}{N})nk}|)^{j(\frac{2\pi}{N})nk}, \quad (k = 0, 1, ..., N-1). \tag{7.2}$$

Where $c[n]$ is the cepstrum. 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 pitch frequency. The pitch period can be found as the number of the coefficient where the peak occurs. Therefore, it seems that cepstrum brings additional information to the evolutionary system, that can be used to perform multi-pitch estimation. One of the advantages of an evolutionary system based on CGP is its ability to choose and evaluate the combinations of inputs to use as long as they do not bring constrains to the system, in this particular case they have to be represented as vectors (arrays). Therefore, to the four inputs used in **CGP-1**, (Figure 6.7), a fifth input was added the **cepstrum**, as shown in Figure 7.2-(e).

Now, the system uses 5 inputs: real part of DFT, imaginary part of DFT, radius of DFT, angle of DFT, and Cepstrum. The cepstrum is calculated using Equation 7.2 applied to the time domain sound signal framed. In Figure 7.2, are depicted 5 graphs, each one represents one of the 5 inputs, being in red the added cepstrum. All the inputs are computed on a sound extracted frame with 4096 time bins, all of them will be real value vectors of 2048 bins. In this particular case, the piano sound is a chord with 3 pitches (70+75+94) and only the first 500 bins are registered in the graphs, for a better and clearer view.

7.2 Onset Detection

Onset Detection is the quest for finding the starting moment of musical notes in an audio signal or according to [Bello et al., 2005],a single instant chosen to mark the temporally extended transient. The transient can be understood as a short-time interval in which a significant energy change occurs in the signal, Figure 7.3. It is an active research subject since note onset detection is commonly used as a first step in high-level music processing tasks.

The purpose of including a block to perform onset detection is to extract from an audio piano sound the frame where the note or chord starts in a more accurate way. The onset detection algorithm used is based on the onset detection algorithm used by [Martins, 2008], with some modifications and improvements. The approach used is based

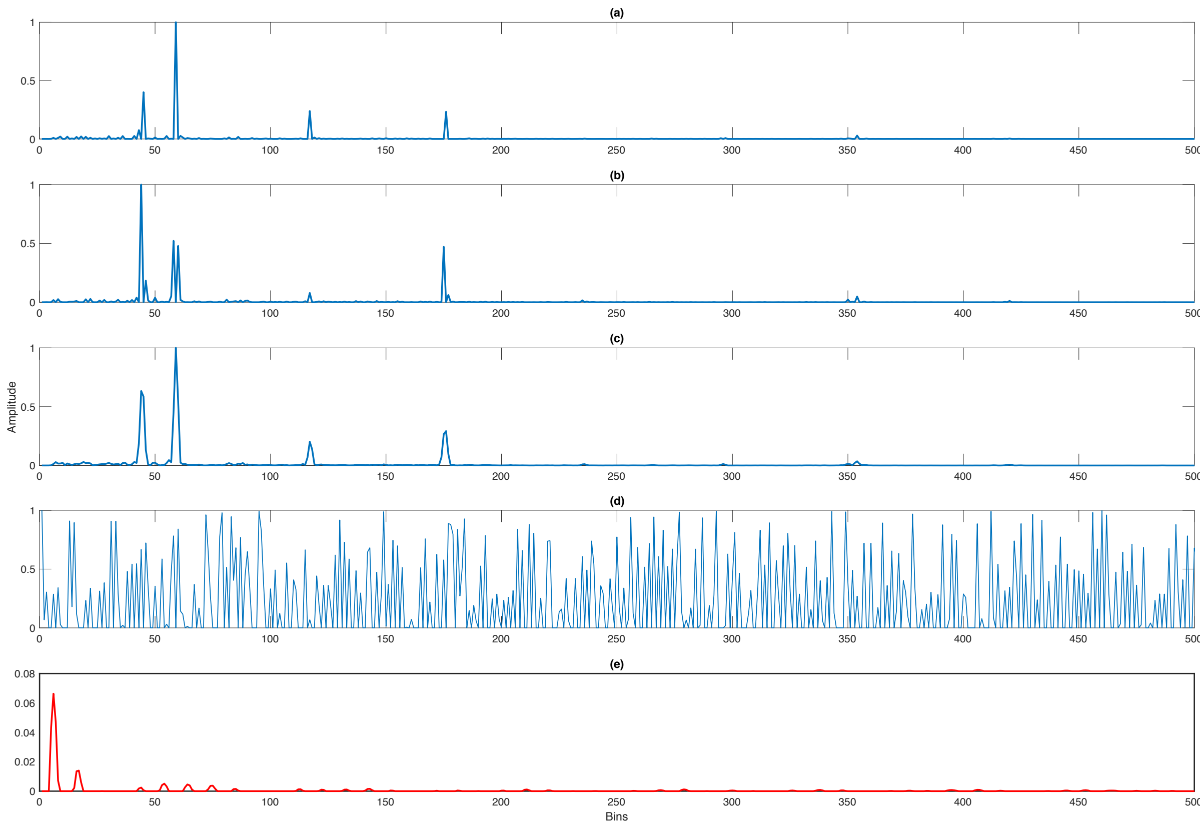


Figure 7.2: System with 5 inputs for a chord with 3 pitches (70+75+94): (**a**) real part of DFT, (**b**) imaginary part of DFT, (**c**) radius of DFT, (**d**) angle of DFT, and (**d**) Cepstrum calculated from the time domain signal. First 500 of 2048 bins of all the 5.

on the Spectral Flux as the onset detection function, defined as:

$$SF[n] = \sum_{k=1}^{\frac{N}{2}} H(|X[n,k]| - |X[n-1,k]|) \tag{7.3}$$

where:

$$H(x) = \frac{x + |x|}{2} \tag{7.4}$$

is the half wave rectifier function, $X(n,k)$ represents the k^{th} bin of the $n^t h$ frame of the short-time Fourier Transform (STFT) of the input $x[n]$. Linear magnitude is used instead of logarithmic. N is the Hamming window size. Experiments performed use a 46 ms frame size (i.e. $N = 2048$ samples, with sampling rate $f_s = 44100$ Hz) and a 11.6 ms hop size ($h = 512$ samples).

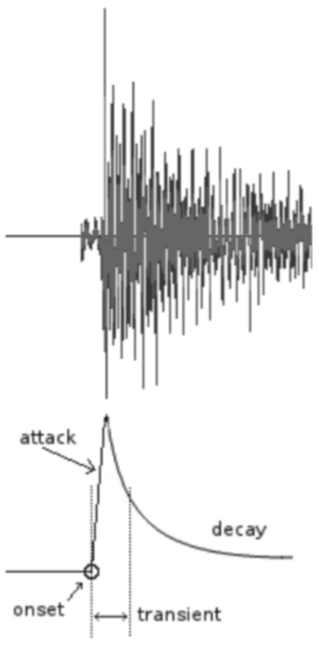


Figure 7.3: Attack, onset and transient in a single note, adapted from [Bello et al., 2005]

As in [Martins, 2008], in order to reduce the false positive rate, the onset detection function $SF(n)$ is smoothed using a Butterworth filter defined by $H(z)$:

$$H(z) = \frac{0.1173 + 0.2347z^{-1} + 0.1174z^{-2}}{1 - 0.8252z^{-1} + 0.2946^{-2}} \tag{7.5}$$

To avoid the phase distortion (which would deviate the time of the detected onsets) the input data is filtered in both forward and backward directions. The result has a precisely zero phase distortion, being the magnitude the square magnitude of the filter response, and the order of the filter the double of the order specified by $H(z)$. We also used an envelope function $EH[p]$, where p represents the discrete time variable of the input signal $x[p]$ [1], the envelope function is calculated over the input time signal $x[p]$, using Hilbert transformation for discrete time signals [Rao et al., 2007].

The onsets are detected using a peak-picking algorithm to find a local maximum. The algorithm is simple and based on 3 conditions:

A peak at instant $t = \frac{nH}{f_s}$ is chosen as an onset if the following conditions are met:

1. $SF[n] \geq SF[k] \; \forall \; k : n - w \leq k \leq n + m$

[1] Input signal in time domain $x[n]$, represented as $x[p]$, because the variable n is now representing the frame number and not the discrete time.

2. $SF[n] > \frac{\sum_{k=n-mw}^{n+w} SF[k]}{mw+w+1} \times thres + \delta$

3. $\frac{\sum_{k=n*h+1}^{n*h+N} EH[k]}{N} > \theta,$

if all three conditions are met, than the peak in instant t is chosen to be an onset instant, where:

$$t = \frac{nH}{f_s} \tag{7.6}$$

The 3 conditions use several parameters for the onset detetction: The $w = 6$ is the window size to achieve the local maxima; $m = 4$ is a multiplier so that the average should be calculated in a broader area before the peak; $thres = 2.0$ is a threshold value relative to the local average that a peak must reach in order to be sufficient prominent to be selected as an onset; and $\delta = 10^{-20}$ is a residual that is combined with $\theta = 0.01$ to avoid false positive detection in silence regions of the signal. All these parameters were adjusted empirically on previously performed tests using a collection of several piano compositions played by different pianos. Figure 7.4 shows an example of a sound sample of 2.1 seconds with 3 onsets detected on frames 29, 90 and 141.

Onset detection is typically used for audio segmentation: since onset detectors infer where musical notes are present in the input audio signal, the input signal can be split in several audio fragments according to the onset information. This is a fundamental process on Automatic Music Transcription algorithms that are based on note tracking. Despite the implemented approach is frame based, the onset detection described process is used during the training stage for data augmentation purposes. Hence, improving the training process. The data augmentation process using the onset detector is explained in Section 7.3.

7.3 Data Augmentation

Data augmentation encompasses a suite of techniques that enhance the size and quality of training datasets [Shorten and Khoshgoftaar, 2019]. In many classification problems, the available data is insufficient to train accurate and robust classifiers. Thus, to alleviate the relative scarcity of the data compared to the number of free parameters of a classifier, one popular approach is data augmentation (DA). Data augmentation consists in transforming the available samples into new samples using label-preserving transformations.

In our classification problem, the amount of data is big enough, however it is not balanced for our multi-pitch approach. For each one of the 61 classifiers to be properly trained, we need to build a balanced dataset with two classes: positive cases and negative cases. Our training data built for each classifier and its training instances are extracted from the MAPS database [Emiya et al., 2010]. This way, we have 61 different classifiers

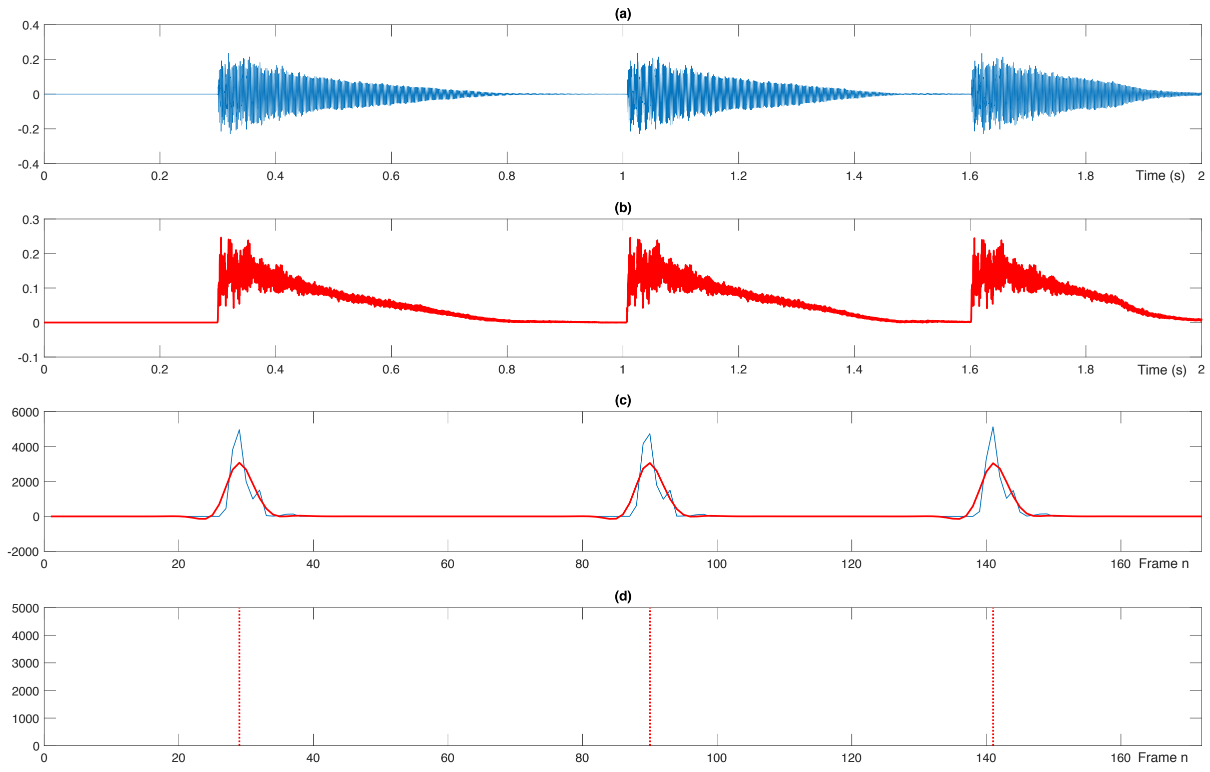


Figure 7.4: Onset detection process steps of a piano signal with 3 onsets: (**a**) original input signal in time domain $x[n]$; (**b**) Hilbert envelope $EH[n]$ of (a); (**c**) the spectral flux $SF[n]$ (blue) , where n represents the frame number and the spectral flux smoothed (red); (**d**) onsets detected and marked in red dashed line.

and 61 different pitches. Therefore, the ratio of positive cases and negative cases in the database for each pitch is $\approx \frac{1}{60}$. However, for proper training, each one of our classifiers requires a balanced dataset, with the same amount of positive and negative cases. This dataset should also be diversified, with the most miscellaneous examples that cover the most range of the pitch space. In our case, we need to enhance the size and quality of the positive subset of the training dataset, in order to be able to use the most number of different negative cases with a good and large distribution in the pitch space (this include multiple pitches combinations) and keep them both balanced in size.

For the data augmentation process, first we apply our onset detector on the input audio signal. Then, based on the inferred onset position (I_{os}), we acquire three different audio fragments translated in time. Each acquired audio fragment has 4096 samples length ($\approx$ 0.93 milliseconds), sampled at frequency 44100 Hz. An audio fragment is acquired at first, starting at the inferred onset time I_{os}. Then, two additional audio frames are acquired from the original signal, starting at $I_{os} - 512$ and $I_{os} + 512$. In practice, we translate the acquired window signal back and forward in time and, this way, we accomplish a data augmentation with factor (3x). This process is illustrated in

Figure 7.5.

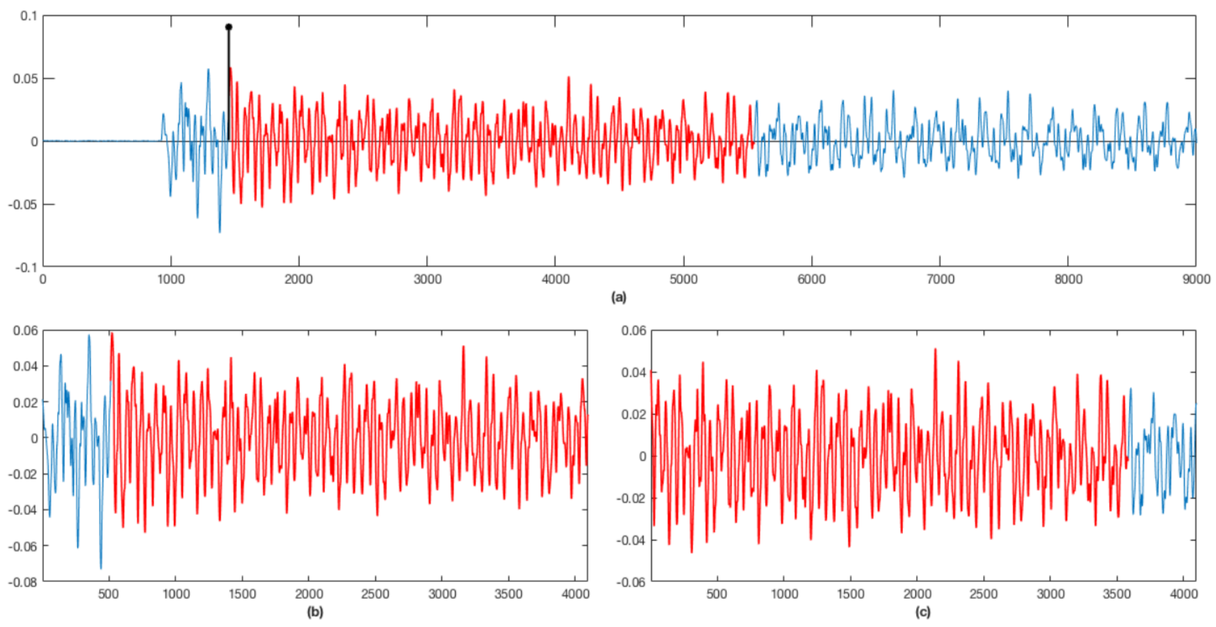


Figure 7.5: Data Augmentation process using time translation: (**a**) in blue, the original piano signal in time domain ($\approx 0.2s$), in black is represented the inferred onset location I_{os}, and in red is the extracted frame, starting at I_{os}; (**b**) audio frame extracted from the original signal, starting at instant $I_{os} - 512$; (**c**) audio frame extracted from the original signal, starting at $I_{os} + 512$.

This innovation in our CGP systems is useful to enlarge the number of positive cases on the training dataset and consequently allows us to enlarge the number of negative original cases maintaining the two classes numbers balanced. This way, we can train the classifiers with a more extensive and diverse negative training dataset. Another important advantage is that, this way, we also force the classifiers to learn and infer not only when the onset detector is accurate but also when it does not estimate the onset time with high precision (delays and advances).

7.4 Function Set

The function set first used was presented in Table 6.1, it contains 24 different functions. After the implementation of the cepstrum as a system input and the use of Hilbert envelope function for the onset detection, it was decided to add also these functions to the function set, first because they were implemented and are easy to add to the system second because they return a vector as output and are commonly used functions in signal processing. Actually the function set was extended with 3 new functions: Cepstrum, Hilbert transform and power.

SPCceps Function that uses only the first input and computes the complex spectrum of the input signal, returns a vector of the same size of the input.

SPEnvelopeHilbert this function has only one argument, a vector, and returns the absolute value of the Hilbert transform as a real vector with the same size.

SPPower this function uses one vector as input and calculates the square of each element of the input vector, it returns also a vector with the same size, in the case of complex numbers it performs the product of each element with its conjugate.

The complete extended function set is detailed in Apendix D using a function look up table.

7.5 Harmonic Mask

The binarization block is the last block before the Fitness function, actually it produces the result of the classification. It is used in the training process to feed the fitness function and it is used in the test stage to produce the classification result of the analysed piano sound, the Harmonic Mask is used to transform the system output vector into a binary number 0 or 1. The idea is similar to the binarization process described in Section 6.9, but using harmonic information to build a mask. In the previous implementation the main goal was to evolve the specific classifier in a manner that, along with the inputs, it could generate an output vector with high values in the classifier fundamental frequency bin. The use of a triangular signal centered in the fundamental frequency with a basis with 3 bins give some room for frequency precision problems. However if the high values of the output of the CGP classifier are placed one bit apart of the exact F0, the system applies a small penalization in value, due to the triangular shape of the signal used to calculate the interception (inner product). This way, small precision deviations are are taking in account in the binarization but have some value penalization because in low frequency pitches the F0 are very close to each other. The implementation of the harmonic mask intends to use more pitch and the signal harmonic information in terms of frequency analysis. Therefore, at the end of each CGP graph (output node), there is a vector of float values, the transformation into a Boolean value is done by applying a spectral mask with harmonic information: each classifier uses its own mask built by the system. The fundamental frequency ($F0$) of the corresponding note being detected by the classifier, is considered the first signal harmonic, thus the Equation 7.7 adapted from Equation 6.9, gives the harmonic frequencies of a piano key or pitch.

$$F(n,p) = n \times 440 \times 2^{\frac{p-49}{12}} Hz, \tag{7.7}$$

where n represents the harmonic number of pitch p. To design the Harmonic Mask vector for pitch p, $M_p[k]$, the system uses the Equation 7.7, to calculate n harmonic frequencies and centers triangles in each harmonic frequency correspondent bin. The final Harmonic Mask is a vector size of output vector in frequency domain with n triangles and zeros otherwise. The amplitude and width of each triangle are configurable system parameters as well as the number of harmonics used. Then the inner product between the output signal and mask is computed:

$$A = \sum_{k=1}^{N} O_{cgp}[k] \times M_p[k], \tag{7.8}$$

where a measures the discrete intersection between the two discrete signals. The rest of the binarization process is equal to the presented in Section 6.9.

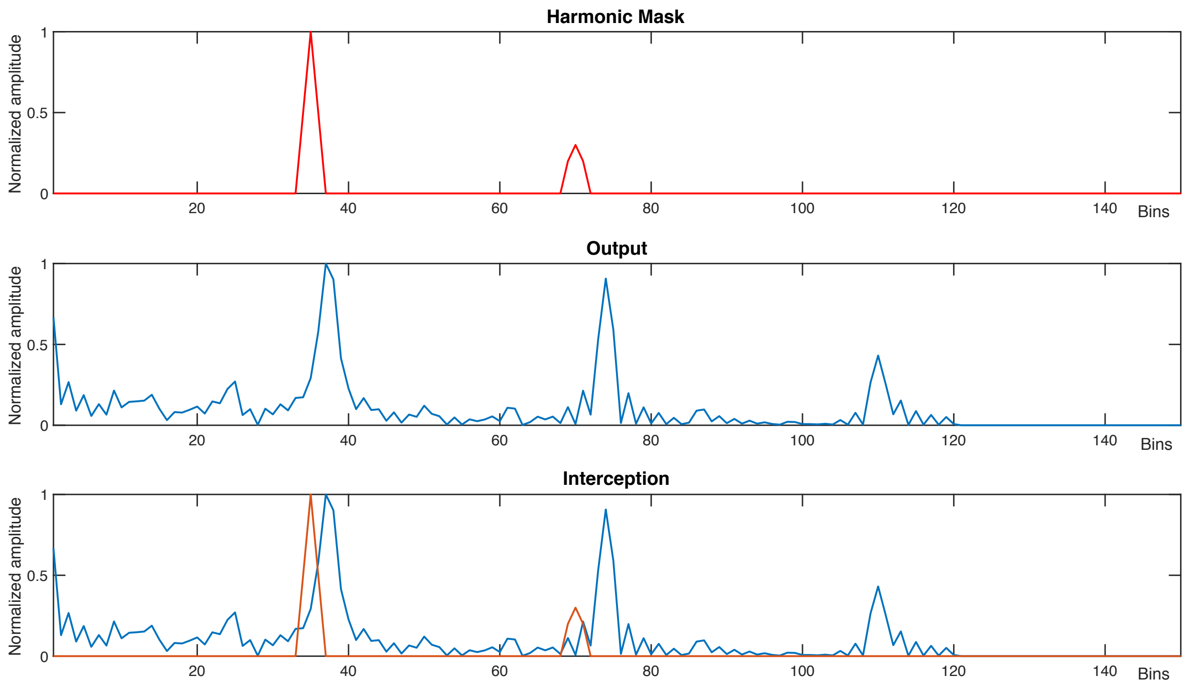


Figure 7.6: Binarization process using the Harmonic Mask for classifier 66: (bottom down) Hrmonic Mask, system output vector, interception

A specific example of the application of the Harmonic Mask in the binarization process is shown in the set of graphs depicted in Figure 7.6. The classifier pitch is 66, and the number of harmonics used are 2, thus for Equation 7.7, the program computes $F(n, 66)$ for $n = 1$ and for $n = 2$. $F(1, 66) \approx 369.99Hz$ and $F(2, 66) \approx 739.99Hz$. The vectors are size of 2048, half of the sound samples in time domain 4096, the frequency precision $\approx 10.7Hz$, thus the frequencies are mapped in the mask vector in position 35 and 70. The triangles configuration acquired from configuration file are: [0.5 1 0.5] and [0.3 0.5 0.3]. Then the output vector O_{cgp} is intercepted with the harmonic mask, M_{66}, generated an interception value A that passes trough a threshold function and generates a Boolean value. The previous example is illustrated in Figure 7.6.

It is possible to observe that the 2 configurable triangles are configured with a base width 3 bins, that fact prevents a possible lack of precision when evaluating the F0 and the correspondent bin, it gives an error margin of one bin. The triangular amplitude rewards precises F0, and small errors are slightly penalized but not completely dismissed. The used configuration also gives more relevance to the fist harmonic placement, however it also takes in account the second harmonic. The changes applied to the Binarization block, in particular the use of the HM, minimize the effect of overlapping of fundamental frequencies or precision errors for pitches with lower frequencies.

7.6 Improved System Overview

Cartesian Genetic Programming builds programs in a form of graphs. To address the MPE problem, the CGP system generates programs in a form of graphs using a set mathematical functions (function set). This way, each each individual will be an evolved graph encoding a complex mathematical expression.

In the final version named **CGP-HM-DA**, each piano note is identified by a CGP evolved classifier (Figure 7.7). This way, for identifying 61 musical notes (from the C2 to the C7), there are 61 evolved classifiers: one for each musical note or piano key. Each one of these classifiers uses several inputs, all of them deriving from the acquired audio signal and return 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. Then, five inputs are computed from the original sound vector using several signal processing techniques. These inputs are then used by the classifiers CGP functions ($F_n(I)$), to accomplish an output vector.

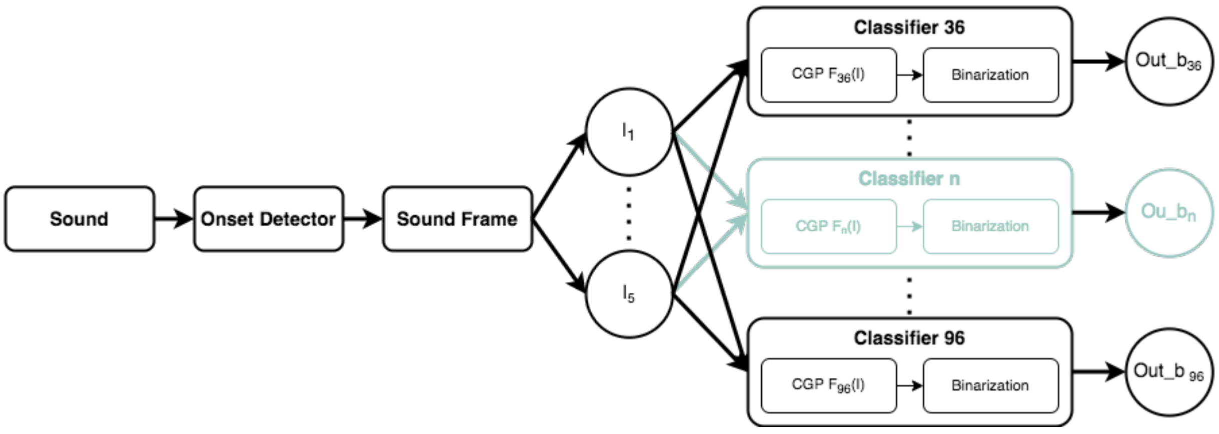


Figure 7.7: Block diagram of the CGP MPE system with multiple classifiers.

In a detailed way the Block diagram represented in Figure 7.7 works as follows: First an onset detector is applied on the input audio signal to infer where there is a start of a musical note; then, an audio frame with 4096 audio samples (93 milliseconds) is extracted, starting on the onset time; the extracted audio fragment is then transformed in 4 different representations on the frequency domain plus a cepstrum, resulting in 5 different inputs ($I_1 \dots I_5$) represented by vectors width; these 5 inputs are feed into the 61 evolved classifiers ($F_n(I)$) that are graphs of mathematical functions from the function set; An output vector is produced $CGPF_n(i)$; finally is applied the binarization process using HM and the binary outputs are computed one for each classifier Out_b_n

7.7 Experiments and Results

To make a detailed study about the quality of the proposed system results (including the 4 versions of it), several experiments were made. The main goal was to evolve 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).

To train and test the system, it was used the MAPS[2] database [Emiya et al., 2010]. It is a piano sound database dedicated to research on Multi-F0 (multi-pitch) estimation and Automatic Music Transcription. It contains piano sound samples of polyphonic and monophonic sounds, usual chords and random chords played by 7 different pianos in different recording conditions and was detailed in Section 6.12.1.

Throughout this section, all the steps performed to test this proposal and their advantages are describe. We started by (i) training and testing some classifiers using the K-fold cross validation method to estimate the skill of our machine learning model on unseen data. Then, (ii) we expanded the experiments to train the CGP system evolving all the 61 classifiers in order to have comparative base with other algorithms. We tested our system composed by the 61 classifiers previously trained with a larger amount of polyphonic piano sounds: 3000 random chords and monophonic notes of 7 different pianos. Additional tasks and tests were made and are described in this section.

7.7.1 System Versions

During Chapter 6, a detailed description was presented about the first system implementation, its features and techniques used. Preliminary tests were made and described to research for the feasibility of the problem of MPE using the Cartesian Genetic Programming. As mentioned, the tests and results were preliminary, they were not exhaustive however they were encouraging and become a crucial starting point for the research. During this chapter (Chapter 7) all the improvements implemented were detailed, not in a chronological order but following the system work flow. Therefore, Table 7.1 presents the 4 system versions with their main features ordered in time.

[2]MAPS stands for MIDI Aligned Piano Sounds and it is available for use under previous request.

Table 7.1: System versions features

Version	Ceps	Tuned Parameters	Ext. FS	HM	DA
CGP-1	$\times$	$\times$	$\times$	$\times$	$\times$
CGP-HM	$\surd$	$\times$	$\times$	$\surd$	$\times$
CGP-HM2	$\surd$	$\surd$	$\surd$	$\surd$	$\times$
CGP-HM-DA	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$

The 5 important features added that contributed for the quality increase of the overall results and for the four tested versions of the system are:

Ceps - Introduction of a fifth system input using cepstrum transform, this feature is presented in Section 7.1.

Tuned Parameters - This was a procedure applied on configurable parameters, the idea was to make a parameters tuning,to improve the results, this was done mainly with the mutation probabilities parameters, a different mutation probability for the threshold and also increase the number of cases during the training stage.

Ext. FS - It is the extended function-set, it was a natural change in the system, when new features where added and new functions were introduced for other purposes. The function set was extended with more functions, the motivation and the added functions are described in Section 7.4

HM - The Harmonic Mask was introduced in the binnarization scheme replacing the original base signal method. This was an important featured implemented in a configurable way, it is presented in Section 7.5

DA - Data Augmentation was the last added feature based on the implementation of a on set detection algorithm. The idea came from other machine learning methods like neural networks, with proved results in classification systems. It allowed a significant enlarge of the training data-set. It was explained in Section 7.3.

The introduction of new features led to new tests and new results, thus the various system implementations, were considered versions and were labeled according to Table 7.1, in the same table is possible to see the main features of each version. Each one of the system versions, along with tests and results produced at least one publication in international conferences or in journals. The fist implementation, **CGP-1** was the genesis of [Inácio et al., 2016]. This implementation as all the others was developed with the **CGP4Matlab** toolbox [Miragaia et al., 2018], published latter after corrections, optimizations and prepared for public use. The **CGP-HM** includes the harmonic mask procedure as well as the cepstrum input, it was described in [Miragaia et al., 2020]. The **CGP-HM2** is a improved version of the previous version, mainly in what concern to system parameters, it was published in [Miragaia et al., 2021b]. Finally, the **CGP-HM-DA**, that includes all the features listed and achieves the most accurate results [Miragaia et al., 2021a].

7.7.2 Validating the Proposed Methodology

The first goal is to validate the proposed methodology based on CGP for MPE of piano sound, using the last system version CGP-HM-DA. The idea is to prove that the methodology works and is capable of infer classification.

A standard procedure for evaluating the performance of classification algorithms is k-fold cross validation [Kohavi et al., 1995, Zhang and Yang, 2015]. The instances in a data set are randomly divided into k disjoint folds with approximately equal size, and every fold is in turn used to test the model induced from the other K-1 folds used for training the model, Figure 7.8 shows a schematic illustration of the K-fold split process with $k = 5$.

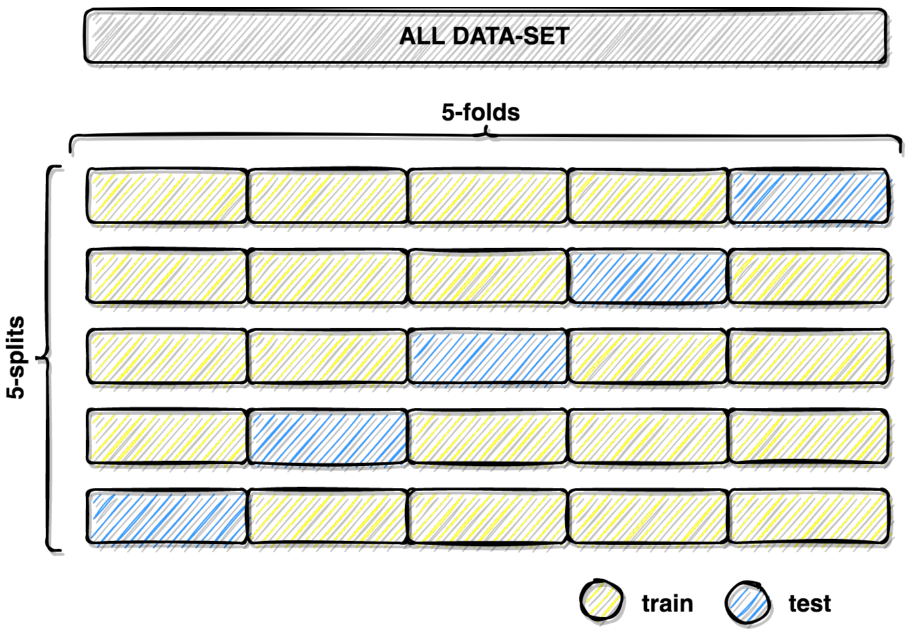


Figure 7.8: K-fold cross validation scheme with $k = 5$: (**gray**) all the data-set, (**yellow**) folds used for training, (**blue**) folds used for test.

Cross-validation is a model validation technique for assessing how the results of a statistical analysis will be able to generalize to a different dataset. It is mainly used in settings where the goal is prediction or classification, and to estimate how accurately a model will perform in practice. To validate the proposed model a 5 fold cross validation method was used, each fold was built using 100 cases, corresponding to a 500 cases dataset for 10 pitches classifiers, randomly chosen. For a specific classifier the dataset was composed by 500 cases, 250 positive cases and 250 negative cases to be balanced in terms of positive and negative cases. All file cases were extracted from MAPS database, where there are monophonic sounds (polyphony 1) and polyphonic sounds, from polyphony 2 to polyphony 6. All the chords were extracted from the RAND subset (random chords subset) which contains random-pitched chords from C2 to B6 uniformly distributed. These chords are composed by random notes without any music rules, like harmonicity and musical consonance. Training the classifiers with random-pitched 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.

Each of the 10 classifiers were trained using 4 folds an were tested with the remaining fold, and the process was repeated for the 5 combinations (5 - splits) of 4 folds (Figure 7.8. For each one of the training and test process were computed 10 runs. The k-fold cross validation and the configurable parameters of our proposed CGP system used, are presented in Table 7.2. The evolutionary process consisted of 10 runs with 10000 generations each, using 200 positive and 200 negative cases for each split combination. The configuration file with the entire parameter values is detailed in Appendix E.

Table 7.2: 5-fold cross validation parameters train/test

Parameter	**Value**
K-folds	5
Data Augmentation	3
Positive Test Cases	250
Negative Test Cases	250
Frame Size	4096
Fitness Initial Threshold	1.5
Outputs	1
Rows	1
Columns	100
Levels Back	100
λ (E.S. 1+λ)	4
Mutation Probability	5%
Threshold Mutation Probability	10%
Harmonic Mask	2
Runs	10
Generations	10000

The k-fold cross validations experiments were conducted with the list of parameter values expressed in Table 7.2. The individual encoding structure is the same used in the first approach; were used one single row with 100 nodes. The “Harmonic Mask” value was empirically set to 2: the CGP system is prepared to use harmonic mask since 1^{st} harmonic to the 5^{th}. The “Data Augmentation” parameter is set to 3, which means that for each initial sound file we generate 3 different ones with time translations forward and backward.

The evolutionary process consisted of 10 runs with 10000 generations each, using the 5 folds dataset split for training and test; were conducted 5 evolutionary processes (train) followed by a test stage, each train stage used 4 folds and was followed by a test stage using the remain unused fold. The classifiers were evaluated using F-measure metric, Equation 6.13. The rest of the parameters remains equal to the previous experiments described in Chapter 6, except the "threshold mutation probability", the system is prepared to an independent value of mutation probability with regard to the gene mutation probability, thus it was set to a higher value 0.1 (10%). The graph depicted in Figure 7.9 shows the 5-fold cross validation results using precision, recall and F-measure. The mean value (μ) for F-measure is 0.93 with 0.91 of precision and 0.93 of recall, these values have a standard deviation (σ) 0.027, 0.025 and 0.025 respectively. These values prove that our CGP based technique is able to learn and infer with a good accuracy rate.

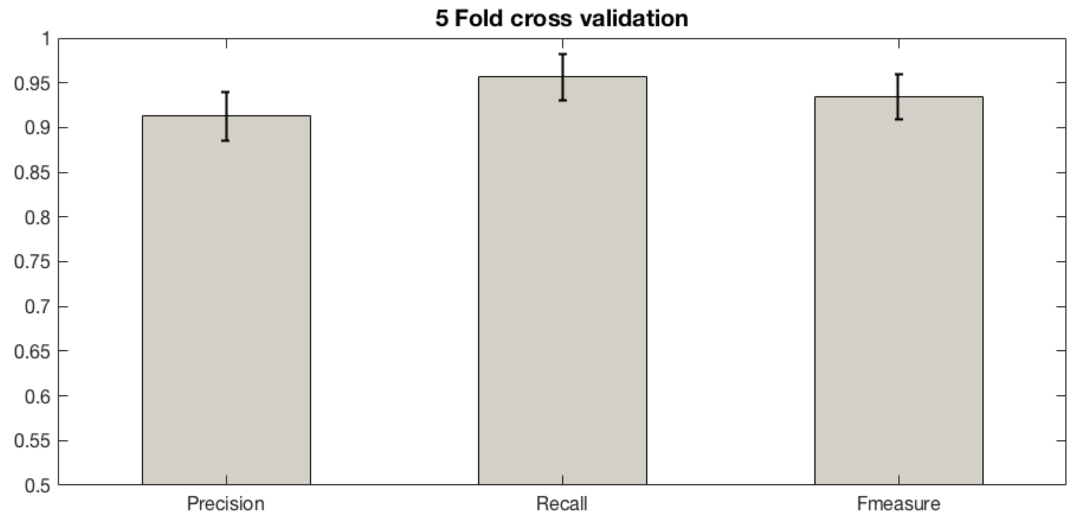


Figure 7.9: Results of 5-fold cross validation for 10 classifiers, using 500 cases dataset, half positive half negative. Precision ($\mu = 0.91, \sigma = 0.027$) Recall ($\mu = 0.95, \sigma = 0.025$) and F-measure ($\mu = 0.93, \sigma = 0.025$)

7.7.3 Classifiers Training

After demonstrating the quality of the machine learning based system with CGP using the k-fold cross validation method for a set of classifiers, it is fundamental to prepare a set of experiments to train all the system classifiers. Therefore, the training stage includes training all of the 61 pitched based classifiers in a way to obtain the best possible results for the testing stage. Thus, a complete training process was made for all the 61 different piano keys to evolve the 61 classifiers. Was used a bigger dataset for the training process, composed of 600 cases of monophonic and polyphonic sounds extracted from the same database used before. The CGP system parameters are identical to those in Table 7.2, except the number of runs per classifier, which we increased to 20. Therefore, each classifier uses a set of 600 cases, 300 positive case and 300 negative, to evolve during 10000 generations.

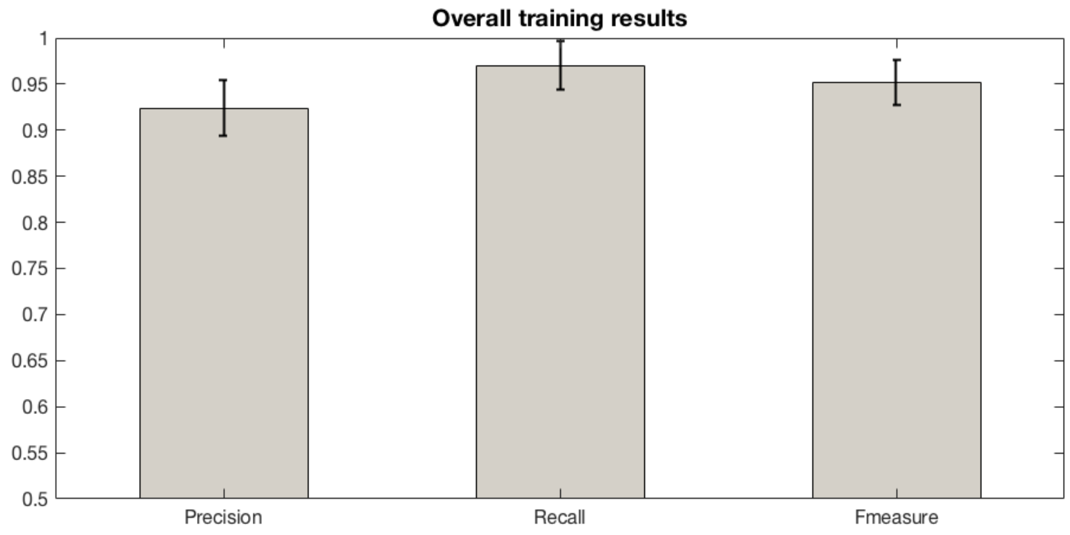


Figure 7.10: Results of the overall training stage for 61 classifiers. Precision ($\mu = 0.92, \sigma = 0.029$) Recall ($\mu = 0.97, \sigma = 0.026$) and F-measure ($\mu = 0.93, \sigma = 0.024$)

In Figure 7.10 the results obtained for the extensive experiment set are shown. The mean F-measure calculated over all the 61 classifiers is 0.95 with $\sigma = 0.024$. These are training results, so as usually we obtained higher values for all the 3 measures computed when compared to test results of unseen data. Typically machine learning classification models should have a bigger training set compared to the test set [Guyon et al., 1997], thus it was important for the quality of the system to train with bigger data-sets however the training process for this model with multiple classifiers has some limitations. It consumes a large amount of time training and that time increases with the amount of cases used (1 run for 1 classifier with 500 cases takes 12 hours), so with the size of the training set. It also requires balanced training sets which means equal number of positive and negative test cases, and the MAPS database does not have that proportion. Although the training stage used a dataset size smaller than the used latter in the test stage due to the resources consumption limitations.

7.7.4 Testing

The cross validation process results gave an idea about the quality of the classifiers. However, the important and comparable with other methods results are those obtained in the test stage with a larger test set comparable with others. The experiments made during the test stage, the classifiers evolved during the training stage were used, to identify the presence of their corresponding pitches in polyphonic audio sounds. We measured the quality of the classifiers by testing them with a different disjoint dataset: the test set. The test set for each classifier consisted in 3,000 piano audio files of chords and single notes extracted from the MAPS database [Emiya et al., 2010], corresponding to almost 5 minutes of music: 3,000 random pitched chords and single notes between C2 (65.406

Hz) to C7 (2093.0 Hz), with polyphony levels ranging from 1 to 6 (includes monophonic sounds). For each sound chord or single note, a $93ms$ frame located after the onset time detected was extracted, sized with 4096. The test results are illustrated in Figure 7.11.

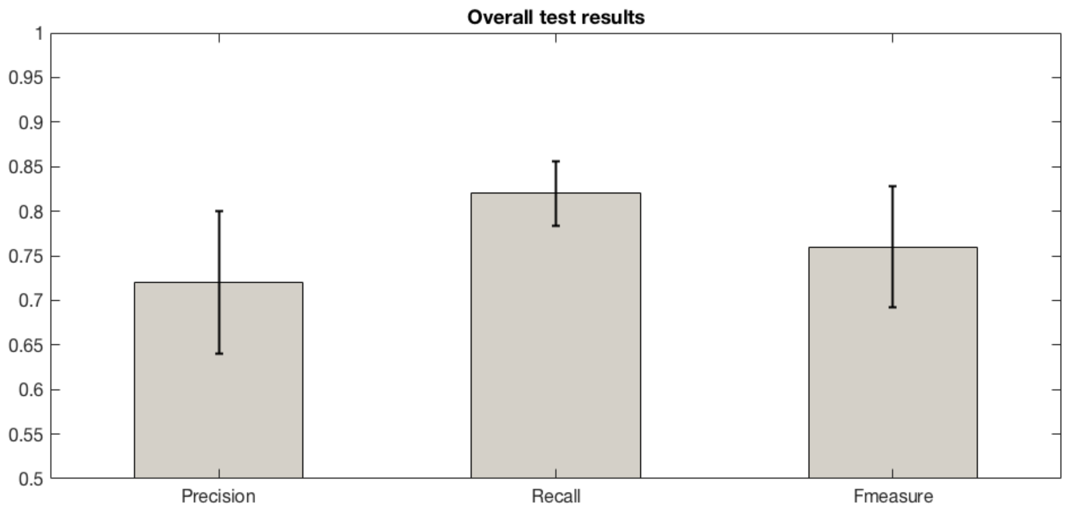


Figure 7.11: Results of the overall testing stage, for 61 classifiers. Precision ($\mu = 0.72, \sigma = 0.08$) Recall ($\mu = 0.82, \sigma = 0.036$) and F-measure ($\mu = 0.764, \sigma = 0.06$)

The system based on the version **CGP-HM-DA** accomplished a final F-measure value of 0.76 (76%). As expected, these are significant worst results, when compared to the values obtained during the training stage, due to the ratio used for the datasets construction because of the referred time and source consumption limitations. However, they are very competitive in the MPE context when compared with other approaches using the same evaluation metric and dataset as will be shown.

Besides the overall test results, a polyphony level experiment was also made. For that purpose the F-measure for each polyphony, from 1 to 6, was also calculated during the test process. The system returns F-measures values in percentage of 77%, 79%, 77%, 73%, 69% and 66% for polyphony[3] 1, 2, 3, 4, 5, and 6, respectively, Figure 7.12 shows detailed results including precision and recall. The results depicted show that the CGP system reaches higher F-measure values for polyphony 2 and 3, and then it decreases until the polyphony 6, with a 13% gap between the highest and lowest polyphony F-measure values. This progressive decrease is related with the number of notes analysed and with the complexity of the problem.

[3]Polyphony 0 - corresponds to silence, easily detected by an analytic power spectrum analysis.

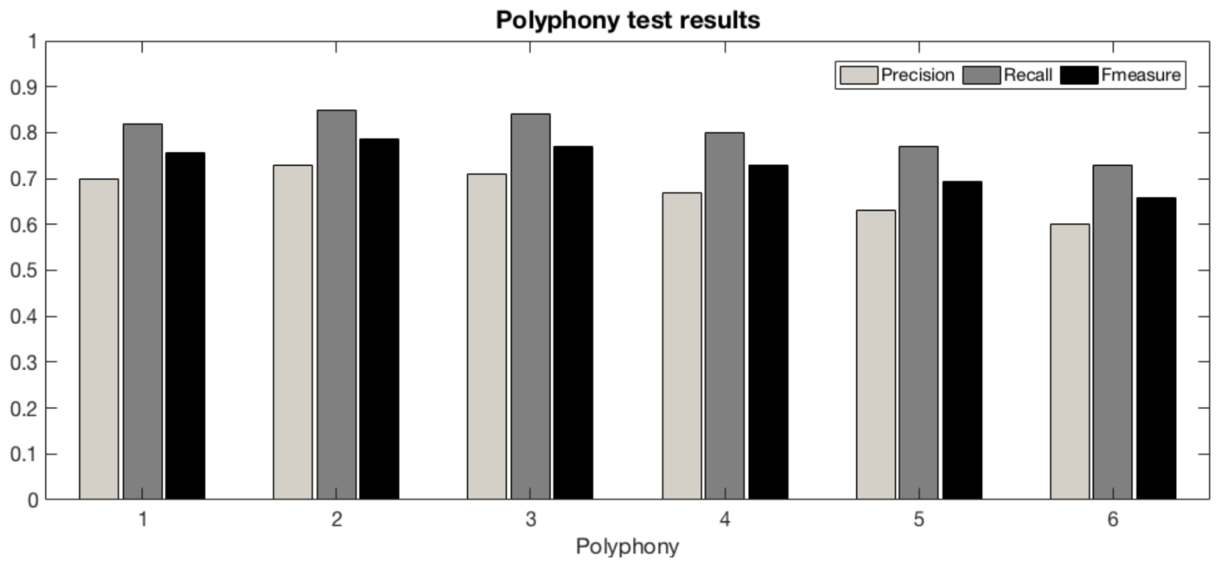


Figure 7.12: Polyphony test results, from polyphony 1 (monophonic) to polyphony 6, results for precision recall and F-measure

Another important issue is related to the piano variety in the database. The dataset includes sounds from seven different pianos. Analysing the results by piano, we have about 4% standard deviation. This means that the system accuracy do not significantly depend on piano differences, recording conditions nor on whether a real piano or a software-based one is used, which suggests robustness for varying production conditions.

7.7.5 System Versions Test Results

In Chapter 6 the first implementation and approach to tackle the problem was introduced, **CGP-1**, it was trained and tested in different dataset conditions, due to its aim and as a starting point for the entire research. As mentioned before, previous versions and several improvements have be made since our initial proposal, until this last version called **CGP-HM-DA**. All the versions were trained and then tested under the same conditions and the test results are expressed in Figure 7.13.

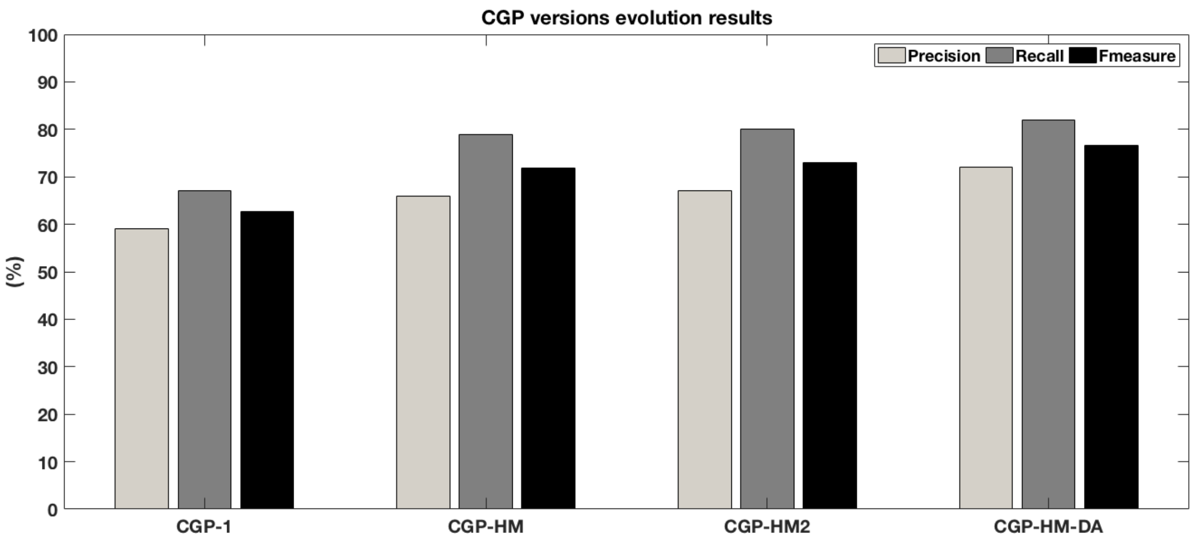


Figure 7.13: CGP MPE system versions: From CGP-1,first approach, to CGP-HM-DA. Results from 62.7% to 76.6%.

Table 7.3: versions results

Version	**Precision**	**Recall**	**F-measure**
CGP-1	59.1	66.9	62.7
CGP-HM	66.2	78.6	71.8
CGP-HM2	67.2	79.9	73
CGP-HM-DA	72.1	82	76.6

Table 7.3 shows F-measure scores for the successive system versions. The improvements and innovations developed, had a direct impact in the quality of the evaluation of the proposed methodology. The starting point was the CGP-1 used and described in [Inácio et al., 2016, Miragaia et al., 2018]. The cepstrum was added and the harmonic mask was developed, CGP-HM in [Miragaia et al., 2020] leading to F-measure of 71.8; later, some functions were added to the function-set and minor improvements were made, related with parameter tuning, CGP-HM2 [Miragaia et al., 2021a], resulting in better F-measure $\approx$ 73%. Finally, the training set was extended using a artificial data augmentation process, CGP-HM-DA, producing the best results for this approach [Miragaia et al., 2021b], which reach F-measure values over than 76%.

7.7.6 Comparing Score to Other MPE Methodologies

There is a lack of Evolutionary approaches on multi-pitch estimation problems in the literature, specially using Genetic Programming. As far as it is public knowledge this methodology based on Cartegian Genetic Programing is unique. Besides, furthermore its architecture with multiple independent classifiers its also peculiar in what concerns to this problem. To make a precise score comparison were used 4 different state of the art algorithms for MPE of piano sound. These were the algorithms that were tested previously by Emiya [Emiya et al., 2009], in the same conditions as this proposal: MPE for random chords using MAPS database including all the pianos, artificial and real.

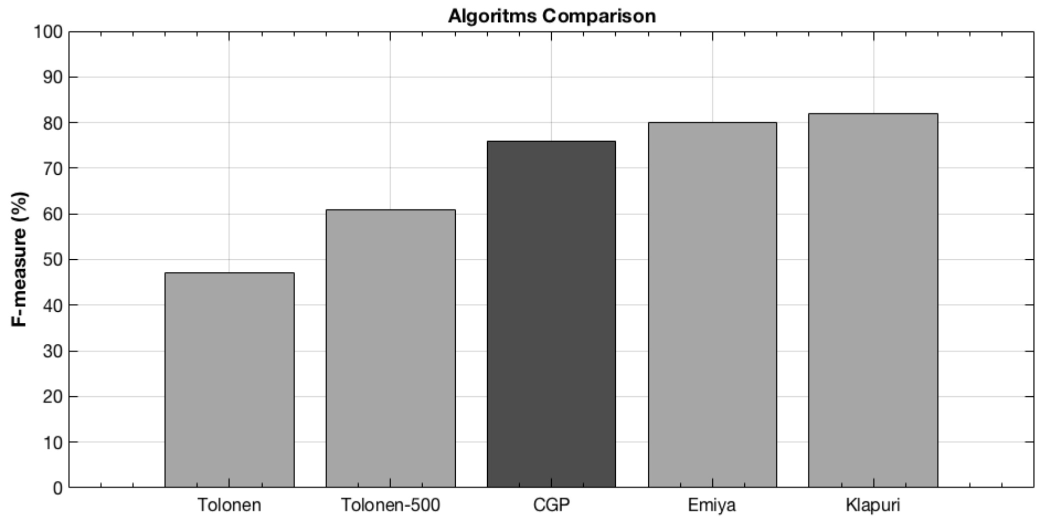


Figure 7.14: F-measure results comparison in (%) with state of the art algorithms: Tolonen, Tolonen-500, Emiya and Klapuri. Our proposal is referenced as CGP and it is the last version of our algorithm, CGP-HM-DA

The 4 algorithms have been tested on the same database for comparison purposesand the metric used is F-measure. The first one is Tolonen's multipitch estimator [Tolonen and Karjalainen, 2000]. As the performance of the algorithm decreases when F0s are greater than 500Hz (C5), the system was additionally tested in restricted conditions – denoted Tolonen-500 – by selecting from the database the subset of sounds composed of notes between C2 and B4 only. The third is Emiya [Emiya et al., 2009]. The fourth one is Klapuri's system [Klapuri, 2003] that has acess to the code, and has been upgraded later [Klapuri, 2008]. The overall results shown in Figure 7.14 demonstrate that the proposed methodology reaches 76% in F-measure, being the third best in quality, only behind Emiya with 80% and Klapuri with 82%. Tolonen reached 47% and Tolonen-500 61%. These results confirm the suitability of the technique employed. Moreover, as it will shown later, this algorithm is able to go beyond piano and has some unique features that stand out. It is important to highlight that there are more recent state of the art algorithms with salient published results, like [Zhang et al., 2020]. However, these recent algorithms were designed and prepared to perform automatic music transcription, most of them based on deep learning methodologies, and they encapsulate the process of multi pitch estimation. Moreover, those were trained and tested in different conditions and thus, a fair comparison can not be made.

7.7.7 White Box Optimization

Many Machine Learning algorithms proved to accomplish successfully their tasks. However, some of them feature “opacity”: they do not provide information about how do they do perform their task in an analytic way. Thus, being labelled as **black box**. The CPG multi-classifier system is completely transparent. It is known that each classifier is composed by a graph of nodes where each node is essentially a mathematical function from a defined function set. Thus, picking any evolved classifier, it is possible to decode it and analyse its readable function graph or mathematical expression. This provides a **white box** analysis. Using classifiers as white box has several advantages. Among them two stand out: one is the possibility to learn from the classifiers obtained, by checking what mathematical functions and inputs are used to make a classification; the second is the possibility to optimize the classifier, tuning the constant values used by the encoded functions.

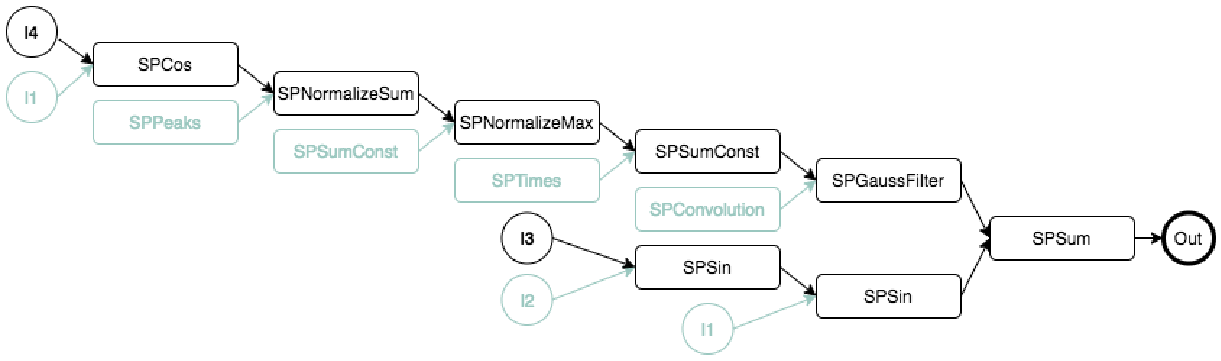


Figure 7.15: Resulting graph after decoding the genotype of classifier 55. The rectangles are functions from the function set, the circles are inputs and the bold one is the output. In grey are the nodes that are not used to calculate the output due to the arity of some functions.

Figure 7.15 shows one of the evolved classifiers, the classifier 55, which as a F-measure of (95%). The decoded graph was obtained decoding the genotype of the classifier presented in Appendix F, besides the genotype it includes the list of active nodes and the final value of the also evolved threshold. Looking at the depicted classifier, the output is computed by a sum (last node) of the resulting expressions: one uses a Gauss filter of two different normalizations of a cosine of I_4 that is the angle of the DFT of the sound signal; and the other branch of the graph uses two consecutive sines of the I_3, which is the radius of number of the DFT. Among all the functions, “SPGaussFilter” executes a sliding window filtering process using a Gaussian function. This function uses a real parameter to design the Gaussian filter. That parameter is the standard deviation (σ). So, a good start to optimize this classifier could be tuning this real constant value. The evolved classifiers are diverse as well as the functions used, although there are almost always functions that uses one or two real parameters. The extended function set is detailed in Appendix-D.

7.7.8 Reaching for the piano and beyond

The main task was to develop a multi-classifier system strong and precise, that could be tested with piano music with accurate results. However it was also important that the developed system was flexible enough to be easily extended to other instruments. After the exhaustive study on MPE for piano music, it was decided to try the MPE of another musical instrument. The study was extended to electric guitar. The idea was to make a prof of concept, try to apply the CGP methodology used in piano sound in another musical instrument. For the study on electric guitar sounds was used a dataset with 400 guitar chords and single notes extracted from guitar database, IDMT-SMT-GUITAR [Kehling et al., 2014]. Seven different guitars in standard tuning were used with varying pick-up settings and different string measures to ensure a sufficient diversification in the field of electric guitars.

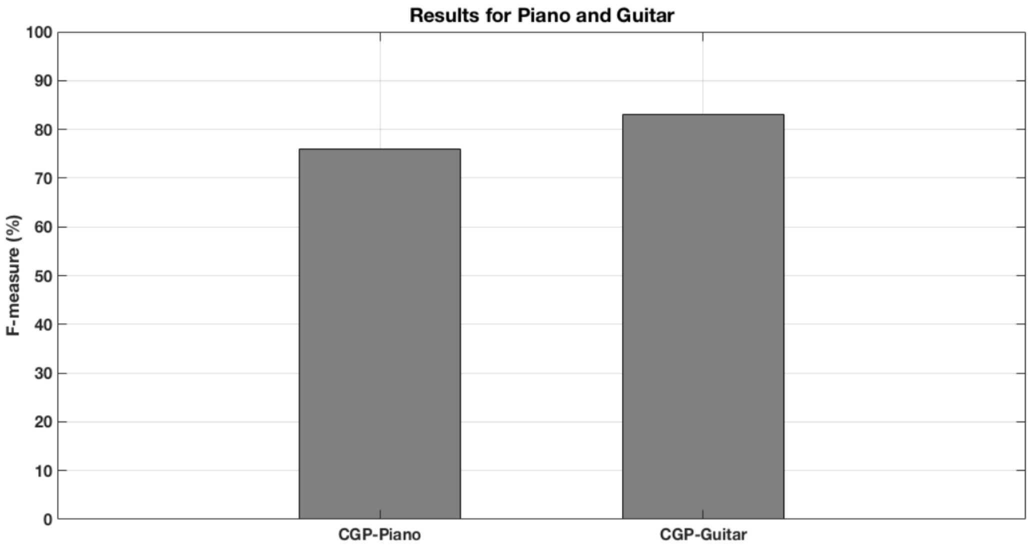


Figure 7.16: F-measure results for MPE on 2 different instruments, piano F-measure 76% and for guitar it rises to 83%.

The procedure adopted was identical to the used for piano: a first stage for training followed by a test stage for unseen data. The results depicted in Figure 7.16, show that the F-measure for guitar reaches the 83%, which are even better than piano results. This way, it is possible to state that the proposed multi-classifier system based CGP has the ability to perform MPE for other musical instruments besides piano, with even better results.

7.7.9 Discussion

To discuss the quality of the CGP proposal it is fundamental to look beyond the score results, based on evaluation metrics. Besides the F-measure results, which have demonstrated the competitiveness of the approach not just for piano, there are a set of features that should be considered when a machine learning technique must be selected for solving a problem. Besides the accuracy, it is important to consider the time consumption, many MPE systems denote poor time performance, therefore the real time feature is one important characteristic. It is also appreciated the capability to perform MPE on another instruments, i.e. to have a system sufficiently generalized to act on another type of musical instruments, the guitar is an example used. The opacity or transparency of the algorithm is another key feature in machine learning approaches, systems with "white boxes" have the ability to show their working process and allow optimization in a non experimental dependent base. Some of these desirable features are summarized in Table 7.4.

Table 7.4: Algorithms main features

Algorithm	Real-Time	Guitar	White Box	Piano FM(%)	Guitar FM(%)
Tolonem	√	x	x	47%	-
Tolonen-500	√	x	x	61%	-
CGP	√	√	√	**76%**	**83%**
Emiya	x	x	x	80%	-
Klapuri	x	x	x	82%	-

This CGP algorithm for MPE is unique in terms of the technique used to address the problem we face. It is the first one which embodies Genetic Programming and a distinctive architecture with multiple classifiers that can work independently and in parallel. As far as it is possible to know, there is no other approach for this kind of problem with a similar multi-classifier architecture. In fact, this architecture is highly parallelizable: since the evolved classifiers are independent, they can run in parallel. This system has been preliminary tested on a 8 core processor: the idea was distribute the classifiers across the processor cores, and due to its parallelization capabilities, it works on real-time. Therefore, as shown in the table, CGP-based approach is the one among the real-time techniques that provides the best F-measure.

Without much effort the CGP system was extended to perform MPE on other instruments with accurate results. We trained and tested our system for electric guitars and we reached an outstanding 83% F-measure. This suggests that this is the only real-time technique developed for piano that may be applied without changes to any instrument. Although the tests have been only successfully conducted to guitar, it is expectable that the results will be positive with other instruments, given that after piano, guitar is another one with higher polyphony degree.

One of the most important features that stands out of the proposed system is the white box method, that allows users and the community to study the solutions provided, by avoiding opacity and allowing future improvements based on what we can learn. Again, this is the only technique among the state of the art techniques featuring white-box optimization. Summarizing, and as shown in Table 7.4, the CGP based system is the only one providing real-time approach, white-box optimization, easily extendable to other instruments, competitive F-measure for piano music and outstanding F-measure values for guitar music.

7.8 Summary

This chapter described all the improvements made to the CGP MPE system first implementation. Among them, the introduction of an Harmonic Mask and the development of an onset detector which leaded to an implementation of a data augmentation process stand up. The improvements were fundamental to increase in a significant way the system accuracy. During this chapter it was also described the various system versions published from **CGP-1** to **CGP-HM-DA**. Finally the detailed set of tests made was explained and a systematized. It was also presented and discussed a feature comparison with other relevant methodologies for the same problem.

Chapter 8

Conclusions and Future Work

The main goal of this dissertation consisted on a research process to apply Cartesian Genetic Programming to the problem of Multi Pitch Estimation of piano notes. The objective was to train 61 classifiers from C1 to C6 piano notes, that could identify their presence in small time-frames of an audio piano signal. This was a process that started with an idea of methodology, but without any tool to accomplish it. Therefore, the first step was to develop a framework or a toolbox capable of addressing the Cartesian Genetic Programming approach to Multi Pitch Estimation, unaware of the problem context. After this first stage, some other small steps were made before tackling the MPE problem for piano. After the development of the CGP4Matlab toolbox, a proof of concept was made for a symbolic regression example of a polynomial mathematical function. The research continued with 2 intermediate steps before the MPE for real audio polyphonic signals: the application of CGP to signals artificially created by mathematical models; and the application of CGP to real audio recordings of monophonic piano signals.

8.1 CGP Toolbox

Multi Pitch Estimation is a complex problem, therefore it was decided to make incremental steps for addressing it. The first and fundamental step was to create, from scratch, a toolbox to help with the codification of the algorithm. Therefore, the first step was defined and it was necessary to choose the aim of the toolbox. Instead of creating a toolbox specially designed for a specific problem, it was decided to create it generic enough to be capable of encoding several different kinds of problems.

The need of a toolbox was clear for some reasons. First of all, there were no toolboxes developed for Matlab for CGP, and it was important to use Matlab because of its singularity and specialization on signal processing techniques. Then, we intended to perform research on MPE problem, which includes many sub-problems, different experiments and parameterisations performed with iterative steps, according to scientific method. Without a toolbox it would take much more time and effort to conduct all the research, i.e much repetitive and redundant code. Another important reason is structural: with the toolbox it was possible to define which code code belongs to the CGP itself, and which code belongs to the pitch estimation problem; this way it was possible to debug and track all the evolutionary process as well as change and analyse in a easy way parameters that modulate that evolution. Finally, with the toolbox developed and available, it was possible to address other problems, particularly image processing and sound processing problems in a more efficient and easier way.

Work with the toolbox is a very easy and intuitive process, it is just needed to provide a small set of configurations, define the system program inputs, define the fitness function and a path to the function-set location. The EA encoded is the $1+\lambda$, and it is possible to choose λ value. Settings such as the mutation probability, number of runs and generations are configurable through a configuration text file. The cartesian representation of CGP (grid of nodes) may assume multiple forms, because the number of rows, columns, levels-back and program outputs are customizable. It is prepared to handle different type of fitness functions: minimization of $f(x)$ and maximization of $f(x)$. It can receive multiple program inputs, of any type. The toolbox is also prepared to receive parameters for each node. Those parameters are encoded in the genotype and can also mutate. Furthermore, it has a useful system of callbacks. This callback system lets us handle multiple events, such as knowing when a new genotype was created, a new solution candidate was found or a run ended. Each callback receives useful information about the event itself: genotype, active nodes, fitness of the current candidate solution, and so on. In summary the CGP4MAtlab toolbox revealed itself as a fundamental tool for the research process and was crucial to achieve the system results and features detailed in this thesis.

8.2 Multi-Pitch Estimation

The preliminary stage of experiments consisted on applying the proposed methodology to artificial signals created with mathematical models and with Gaussian noise addition. Sine functions, square waves and sawtooth waves were used. All the evolved classifiers for pitch estimation estimated all the correct fundamental pitches on true positive signals and correctly identified all the true negative cases. It was very promising and a strong starting point for further steps. Then, the system was designed and evolved for real piano polyphonic signals, leading to the first tested version CGP-1, which presented the first published results. This required a new set of parameters as the implementation of a new real signal functions for the new function set. This first set of results was the basis for the rest of the research, and the results had room to increase, with some innovations. The use of the Harmonic Mask, the extended function set, the use of an additional input based on cepstrum as well as the use of an independent higher mutation probability for the fitness threshold contributed for two more version with published results, named CGP-HM and CGP-HM2 respectively. Finally, we proposed the most recent version of the multi-pitch estimation system for piano music based on Cartesian Genetic Programming using an harmonic mask and improved with a data augmentation process for a better training stage. An onset detection analytic system was also introduced to aid the working process.

The last version's results were compiled, as well as the methodology used, and were also published.

The main disadvantage of our proposal is the fact that it requires evolving the 61 classifiers, so that the system is able to employ those classifiers to perform the MPE, and the training (evolutionary) process takes time: each single classifier takes one day to train (10 runs). However, our technique produces competitive results for any kind of piano sounds without the need of harmony rules for chords construction. This way, it is more generalized and works with random notes, which could be of interest for contemporary music. Moreover, we have also shown that the methodology can be applied without changes to other polyphonic instruments, such as guitar.

The CGP-HM-DA approach based on a graphs of nodes with functions extracted from a dataset can be easily decoded once the classifiers have been evolved for a given instrument: the classifiers are made up of mathematical expressions with inputs, internal functions and outputs, which provides "white box" optimization.

Results are shown and compared with state of the art algorithms and we demonstrate the feasibility of the approach: the technique is the only one providing simultaneously (i) competitive results (F-measure) together with (ii) real-time capabilities thanks to their intrinsic parallel nature; (iii) white-box optimization; (iv) and can be directly applied to other polyphonic instruments with outstanding results (F-measure).

8.3 Future Work

We consider that this thesis enables a small step in the world of Multi Pitch Estimation, introducing a new technique in the field of Evolutionary algorithms – the Cartesian Genetic Programming –, but more can be done. Future work may include some of the following items:

- Preparing the toolbox for recent developments in the field of CGP, such as "Crossover" and Modular CGP.

- Prepare the toolbox for GPU Parallel Processing, as GPUs offer significant speed boost at a time when the CPU performance increase has slowed down over the past few years. This feature is important in the field of Data Science, which involves processing very large datasets (in the form of matrices/vectors) efficiently. The SIMD design, or Single Instruction/Multiple Data, means that GPU computing can process multiple data with a single instruction, as is the case for matrix.

- In the proposed system we used five different inputs using FFT and cepstrum, add more program inputs using another type of transforms such as "Constant-Q transform" increases the inputs variety for a better training stage.

- The extension of the function set can also be of interest; there are several numeric functions for peak detection or for filtering processes that may be useful and add variety to the function set.
- Another suggestion would be experiment different binarization methods; we only used the output signal interception with a Mask, associated to a threshold process, however another similarity methods can be used, such as Correlation, or distance between vectors.
- Future work will focus on demonstrating the capabilities of the proposed system on other types of pitched instruments, namely: woodwind and string ensemble.
- It will be interesting with the aid of more computational power to use different and larger grids of nodes for the CGP system; it would allow more complex and longer chains of mathematical functions.
- Also, with the increase of computational resources, the classifiers should be trained with huge data-sets in a similar way to deep learning processes; bigger datasets have more variety of cases and usually allow system to train mpre adequately.
- Use this MPE approach to built a more generic automatic music transcription system.

It is fair to state that this dissertation is a valid contribution to the study of the Multi Pitch Estimation problem, suggesting one more valid and innovating technique to the existing ones.

Bibliography

[Alonso et al., 2005] Alonso, M., Richard, G., and David, B. (2005). Extracting note onsets from musical recordings. In *2005 IEEE International Conference on Multimedia and Expo*, pages 4 pp.–. [cited at p. 55]

[Argenti et al., 2010] Argenti, F., Nesi, P., and Pantaleo, G. (2010). Automatic transcription of polyphonic music based on the constant-q bispectral analysis. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(6):1610–1630. [cited at p. 55]

[Arora and Behera, 2015] Arora, V. and Behera, L. (2015). Multiple f0 estimation and source clustering of polyphonic music audio using plca and hmrfs. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(2):278–287. [cited at p. 58]

[Association, 2008] Association, M. M. (1999/2008). *Complete MIDI 1.0 Detailed Specification.* [cited at p. 14]

[Beauchamp et al., 2012] Beauchamp, J., Maher, R., and Brown, R. (2012). Detection of musical pitch from recorded solo performances. *Watermark*, 1. [cited at p. 50]

[Bello et al., 2005] Bello, J., Daudet, L., Abdallah, S., Duxbury, C., Davies, M., and Sandler, M. (2005). A tutorial on onset detection in music signals. *Speech and Audio Processing, IEEE Transactions on*, 13(5):1035–1047. [cited at p. XVII, 49, 121, 123]

[Bello et al., 2002] Bello, J., Daudet, L., and Sandler, M. (2002). Time-domain polyphonic transcription using self-generating databases. In *112th Convention of the Audio Engineering Society*, Munich, Germany. [cited at p. 58]

[Bello et al., 2006] Bello, J. P., Daudet, L., and Sandler, M. B. (2006). Automatic piano transcription using frequency and time-domain information. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(6):2242–2251. [cited at p. 54]

[Bello et al., 2000] Bello, J. P., Monti, G., Sandler, M. B., et al. (2000). Techniques for automatic music transcription. In *ISMIR*. [cited at p. 55]

[Bello and Sandler, 2000] Bello, J. P. and Sandler, M. (2000). Blackboard system and top-down processing for the transcription of simple polyphonic music. In *In Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFx-00*, pages 7–9. [cited at p. 55]

[Benetos and Dixon, 2011] Benetos, E. and Dixon, S. (2011). Multiple-instrument polyphonic music transcription using a convolutive probabilistic model. *8th Sound and Music Computing Conf*, page 19–24. [cited at p. 58]

[Benetos et al., 2018] Benetos, E., Dixon, S., Duan, Z., and Ewert, S. (2018). Automatic music transcription: An overview. *IEEE Signal Processing Magazine*, 36(1):20–30. [cited at p. 51, 57]

[Benetos et al., 2013] Benetos, E., Dixon, S., Giannoulis, D., Kirchhoff, H., and Klapuri, A. (2013). Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 41(3):407–434. [cited at p. 52]

[Bertin et al., 2010] Bertin, N., Badeau, R., and Vincent, E. (2010). Enforcing harmonicity and smoothness in bayesian non-negative matrix factorization applied to polyphonic music transcription. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):538–549. [cited at p. 58]

[Böck and Schedl, 2012] Böck, S. and Schedl, M. (2012). Polyphonic piano note transcription with recurrent neural networks. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 121–124. IEEE. [cited at p. 59]

[Bogert, 1963] Bogert, B. P. (1963). The quefrency alanysis of time series for echoes; cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking. *Time series analysis*, pages 209–243. [cited at p. 39]

[Brown, 1992] Brown, J. (1992). Musical fundamental frequency tracking using a pattern recognition method. *The Journal of the Acoustical Society of America*, 92:1394. [cited at p. 42]

[Casey et al., 2008] Casey, M. A., Veltkamp, R., Goto, M., Leman, M., Rhodes, C., and Slaney, M. (2008). Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696. [cited at p. 89]

[Chakravorty, 2018] Chakravorty, P. (2018). What is a signal? [lecture notes]. *IEEE Signal Processing Magazine*, 35(5):175–177. [cited at p. 16]

[Chinchor, 1992] Chinchor, N. (1992). Muc-4 evaluation metrics in proc. of the fourth message understanding conference 22–29. [cited at p. 111]

[Clegg et al., 2007] Clegg, J., Walker, J. A., and Miller, J. F. (2007). A new crossover technique for cartesian genetic programming. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1580–1587. [cited at p. 78]

[Conklin Jr, 1999] Conklin Jr, H. (1999). Generation of partials due to nonlinear mixing in a stringed instrument. *The Journal of the Acoustical Society of America*, 105:536. [cited at p. 49]

[Cont, 2006] Cont, A. (2006). Realtime multiple pitch observation using sparse non-negative constraints. In *Proc. Int. Conf. on Music Information Retrieval (ISMIR)*, pages 206–212. [cited at p. 57]

[Cooley et al., 1967] Cooley, J. W., Lewis, P. A., and Welch, P. D. (1967). Historical notes on the fast fourier transform. *Proceedings of the IEEE*, 55(10):1675–1677. [cited at p. 26, 31]

[Corkill, 1991] Corkill, D. D. (1991). Blackboard systems. *AI Expert*, 6:40–47. [cited at p. 54]

[Darwin, 1859] Darwin, C. (1859). *On the Origin of the Species by Natural Selection*. Murray, London, UK. [cited at p. 63]

[Daudet, 2004] Daudet, L. (2004). Sparse and structured decompositions of audio signals in overcomplete spaces. In *Proc. Int. Conference on Digital Audio Effects (DAFx), Naples, Italy*, pages 22–26. [cited at p. 49]

[Davy and Godsill, 2003] Davy, M. and Godsill, S. (2003). Bayesian harmonic models for musical signal analysis. In Bernardo, J. M., editor, *Bayesian Statistics VII*. Oxford University Press. [cited at p. 56]

[Davy et al., 2006] Davy, M., Godsill, S., and Idier, J. (2006). Bayesian analysis of polyphonic western tonal music. *The Journal of the Acoustical Society of America*, 119(4):2498–2517. [cited at p. 56]

[De Cheveigné and Kawahara, 2002] De Cheveigné, A. and Kawahara, H. (2002). Yin, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111:1917. [cited at p. 39]

[De Jong, 1975] De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, Ann Arbor, MI, USA. AAI7609381. [cited at p. 66]

[de Vega, 2001] de Vega, F. F. (2001). *Distributed genetic programming models with application to logic synthesis on FPGAs.* PhD thesis, PhD thesis, University of Extremadura, 2001.† FFernandez-de-Vega .... [cited at p. 63]

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38. [cited at p. 56]

[Dictionary, 2002] Dictionary, C. O. E. (2002). *Concise Oxford English Dictionary - Thumb Index Edition.* Oxford University Press, tenth edition. [cited at p. 13]

[Donoho, 2006] Donoho, D. (2006). For most large underdetermined systems of linear equations the minimal l1-norm solution is also the sparsest solution. *Communications on pure and applied mathematics*, 59(6):797–829. [cited at p. 58]

[Duan et al., 2010] Duan, Z., Pardo, B., and Zhang, C. (2010). Multiple fundamental frequency estimation by modeling spectral peaks and non-peak regions. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(8):2121–2133. [cited at p. 56]

[Duan et al., 2008] Duan, Z., Zhang, Y., Zhang, C., and Shi, Z. (2008). Unsupervised single-channel music source separation by average harmonic structure modeling. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(4):766–778. [cited at p. 56]

[Duifhuis et al., 1982] Duifhuis, H., Willems, L., and Sluyter, R. (1982). Measurement of pitch in speech: An implementation of goldsteinâ"s theory of pitch perception. *The Journal of the Acoustical Society of America*, 71:1568. [cited at p. 42]

[Eigen, 1973] Eigen, M. (1973). *Ingo Rechenberg Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologishen Evolution.* mit einem Nachwort von Manfred Eigen, Friedrich Frommann Verlag, Struttgart-Bad Cannstatt. [cited at p. 107]

[Ellis, 1996] Ellis, D. P. W. (1996). *Prediction-driven Computational Auditory Scene Analysis.* PhD thesis, Cambridge, MA, USA. AAI0597425. [cited at p. 55]

[Emiya, 2007] Emiya, V. (2007). Multipitch estimation and tracking of inharmonic sounds in colored noise. [cited at p. 55]

[Emiya, 2008] Emiya, V. (2008). *Transcription automatique de la musique de piano.* PhD thesis. Thèse de doctorat dirigée par David, Bertrand et Badeau, Roland Signal et images Paris, ENST 2008. [cited at p. 89]

[Emiya et al., 2007a] Emiya, V., Badeau, R., and David, B. (2007a). Multipitch estimation of quasi-harmonic sounds in colored noise. In *10th Int. Conf. on Digital Audio Effects (DAFx-07)*, Bordeaux, France. [cited at p. 55, 56]

[Emiya et al., 2009] Emiya, V., Badeau, R., and David, B. (2009). Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1643–1654. [cited at p. 56, 139]

[Emiya et al., 2010] Emiya, V., Bertin, N., David, B., and Badeau, R. (2010). Maps-a piano database for multipitch estimation and automatic transcription of music. [cited at p. 52, 114, 124, 130, 135]

[Emiya et al., 2007b] Emiya, V., David, B., and Badeau, R. (2007b). A parametric method for pitch estimation of piano tones. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 1, pages I–249. IEEE. [cited at p. 54]

[Every and Szymanski, 2004] Every, M. and Szymanski, J. (2004). A spectral-filtering approach to music signal separation. pages 197–200. [cited at p. 47]

[Ewert and Sandler, 2016] Ewert, S. and Sandler, M. (2016). Piano transcription in the studio using an extensible alternating directions framework. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(11):1983–1997. [cited at p. 59]

[Fletcher and Rossing, 1998] Fletcher, N. H. and Rossing, T. D. (1998). *The physics of musical instruments / Neville H. Fletcher, Thomas D. Rossing.* Springer, New York ; London :, 2nd ed. edition. [cited at p. 49]

[Fogel, 1992] Fogel, D. B. (1992). *Evolving artificial intelligence.* PhD thesis, La Jolla, CA, USA. UMI Order No. GAX93-03240. [cited at p. 67]

[Fogel et al., 1966] Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution.* John Wiley, New York, USA. [cited at p. 67]

[Fritts, 2006] Fritts, L. (2006). Available: http:// theremin.music.uiowa.edu/mis.html last accessed: 2016-06-20 University of Iowa Musical Instrument Samples. http://theremin.music.uiowa.edu/. [cited at p. 52]

[Gandomi et al., 2015] Gandomi, A. H., Alavi, A. H., and Ryan, C. (2015). *Handbook of genetic programming applications.* Springer. [cited at p. 61]

[Godsill and Davy, 2002] Godsill, S. and Davy, M. (2002). Bayesian harmonic models for musical pitch estimation and analysis. *Acoustics, Speech, and Signal Processing, 2002. Proceedings. (ICASSP '02). IEEE International Conference on*, 2:1769–1772. [cited at p. 56]

[Goldberg and Holland, 1988] Goldberg, D. E. and Holland, J. H. (1988). Genetic algorithms and machine learning. [cited at p. 91]

[Goldman and Punch, 2015] Goldman, B. W. and Punch, W. F. (2015). Analysis of cartesian genetic programming's evolutionary mechanisms. *IEEE Transactions on Evolutionary Computation*, 19(3):359–373. [cited at p. 107]

[Goldstein, 1973] Goldstein, J. (1973). An optimum processor theory for the central formation of the pitch of complex tones. *The Journal of the Acoustical Society of America*, 54(6):1496–1516. [cited at p. 42]

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning.* MIT press. [cited at p. 59]

[Goto, 2004] Goto, M. (2004). A real-time music-scene-description system: predominant-F0 estimation for detecting melody and bass lines in real-world audio signals. *Speech Communication*, 43(4):311–329. [cited at p. 56]

[Goto et al., 2002] Goto, M., Hashiguchi, H., Nishimura, T., and Oka, R. (2002). Rwc music database: Popular, classical and jazz music databases. In *Proceedings of the Third International Conference on Music Information Retrieval*, pages 287–288, Paris, France. `http://ismir2002.ismir.net/proceedings/03-SP04-1.pdf`. [cited at p. 52]

[Guyon et al., 1997] Guyon, I. et al. (1997). A scaling law for the validation-set training-set size ratio. *AT&T Bell Laboratories*, 1(11). [cited at p. 135]

[H. Viste and G. Evangelista, 2002] H. Viste and G. Evangelista (2002). An Extension for Source Separation Techniques Avoiding Beats. In *Proc. of Digital Audio Effects Conference (DAFx '02)*, Hamburg, Germany. [cited at p. 47]

[Hainsworth, 2003] Hainsworth, S. W. (2003). *Techniques for the automated analysis of musical audio.* PhD thesis, University of Cambridge. [cited at p. 61]

[Hansen et al., 2015] Hansen, N., Arnold, D. V., and Auger, A. (2015). *Evolution Strategies*, pages 871–898. Springer Berlin Heidelberg, Berlin, Heidelberg. [cited at p. 77]

[Harding et al., 2013] Harding, S., Leitner, J., and Schmidhuber, J. (2013). Cartesian genetic programming for image processing. In *Genetic Programming Theory and Practice X*, pages 31–44. Springer. [cited at p. 61, 79, 91]

[Harris, 1978] Harris, F. J. (1978). On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83. [cited at p. 33]

[Hartmann, 1997] Hartmann, W. (1997). *Signals, Sound, and Sensation.* Modern Acoustics and Signal Processing. American Inst. of Physics. [cited at p. 35]

[Hawthorne et al., 2017] Hawthorne, C., Elsen, E., Song, J., Roberts, A., Simon, I., Raffel, C., Engel, J., Oore, S., and Eck, D. (2017). Onsets and frames: Dual-objective piano transcription. *arXiv preprint arXiv:1710.11153.* [cited at p. 60]

[Hess, 1983] Hess, W. (1983). *Pitch determination of speech signals: algorithms and devices.* Springer series in information sciences. Springer-Verlag. [cited at p. 38, 39]

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, USA. [cited at p. 64, 66]

[Hynek Hermansky, 1993] Hynek Hermansky, Nelson Morgan, H.-G. H. (1993). Recognition of speech in additive and convolutional noise based on rasta spectral processing. *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2:83–86. [cited at p. 53]

[Inácio et al., 2016] Inácio, T., Miragaia, R., Reis, G., Grilo, C., and Fernandéz, F. (2016). Cartesian genetic programming applied to pitch estimation of piano notes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE. [cited at p. 131, 138]

[Kameoka et al., 2007] Kameoka, H., Nishimoto, T., and Sagayama, S. (2007). A multipitch analyzer based on harmonic temporal structured clustering. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(3):982–994. [cited at p. 56]

[Kehling et al., 2014] Kehling, C., Abeßer, J., Dittmar, C., and Schuller, G. (2014). Automatic tablature transcription of electric guitar recordings by estimation of score-and instrument-related parameters. In *DAFx*, pages 219–226. [cited at p. 141]

[Kelz et al., 2019] Kelz, R., Böck, S., and Widmer, G. (2019). Deep polyphonic adsr piano note transcription. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 246–250. IEEE. [cited at p. 60]

[Kelz et al., 2016] Kelz, R., Dorfer, M., Korzeniowski, F., Böck, S., Arzt, A., and Widmer, G. (2016). On the potential of simple framewise approaches to piano transcription. *arXiv preprint arXiv:1612.05153.* [cited at p. 60]

[Klapuri, 1998] Klapuri, A. (1998). Number theoretical means of resolving a mixture of several harmonic sounds. In *Proceedings of the European Signal Processing Conference*, page 400. [cited at p. 46]

[Klapuri, 2004a] Klapuri, A. (2004a). *Signal processing methods for the automatic transcription of music.* PhD thesis, Tampere University of Technology Finland. [cited at p. XV, 40, 43]

[Klapuri, 2005] Klapuri, A. (2005). A perceptually motivated multiple-f0 estimation method. In *in Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 291–294. [cited at p. 53, 61]

[Klapuri, 2006] Klapuri, A. (2006). Multiple fundamental frequency estimation by summing harmonic amplitudes. In *in ISMIR*, pages 216–221. [cited at p. 53]

[Klapuri, 2008] Klapuri, A. (2008). Multipitch analysis of polyphonic music and speech signals using an auditory model. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):255–266. [cited at p. 139]

[Klapuri and Davy, 2006] Klapuri, A. and Davy, M., editors (2006). Springer, New York. [cited at p. 36]

[Klapuri, 2003] Klapuri, A. P. (2003). Multiple fundamental frequency estimation based on harmonicity and spectral smoothness. *IEEE Transactions on Speech and Audio Processing*, 11(6):804–816. [cited at p. 53, 54, 139]

[Klapuri, 2004b] Klapuri, A. P. (2004b). Automatic music transcription as we know it today. *Journal of New Music Research*, 33(3):269–282. [cited at p. 14]

[Klapuri and Astola, 2002] Klapuri, A. P. and Astola, J. T. (2002). Efficient calculation of a physiologically-motivated representation for sound. In *Digital Signal Processing, 2002. DSP 2002. 2002 14th International Conference on*, volume 2, pages 587–590. IEEE. [cited at p. 43]

[Kohavi et al., 1995] Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada. [cited at p. 132]

[Koretz and Tabrikian, 2011] Koretz, A. and Tabrikian, J. (2011). Maximum a posteriori probability multiple-pitch tracking using the harmonic model. *Trans. Audio, Speech and Lang. Proc.*, 19(7):2210–2221. [cited at p. 56]

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. A Bradford Book, 1 edition. [cited at p. 67, 70]

[Koza, 1994] Koza, J. R. (1994). Genetic programming ii: Automatic discovery of reusable subprograms. *Cambridge, MA, USA*. [cited at p. 67, 70]

[Lee et al., 2012] Lee, C.-T., Yang, Y.-H., and Chen, H. H. (2012). Multipitch estimation of piano music by exemplar-based sparse representation. *IEEE Transactions on Multimedia*, 14(3):608–618. [cited at p. XVI, 90]

[Lee et al., 2010] Lee, C.-T., Yang, Y.-H., Lin, K.-S., and Chen, H. (2010). Multiple fundamental frequency estimation of piano signals via sparse representation of fourier coefficients. In *Proceedings of the Fourth Music Information Retrieval Evaluation eXchange (MIREX 2010)*, Utrecht. [cited at p. 58]

[Luke, 2011] Luke, S. (2011). Essentials of metaheuristics. lulu, 2009. *Available for free at http://cs. gmu. edu/sean/book/metaheuristics/. There is no corresponding record for this reference.* [cited at p. 67, 70]

[Marolt, 2004] Marolt, M. (2004). A connectionist approach to automatic transcription of polyphonic piano music. *IEEE Transactions on Multimedia*, 6(3):439–449. [cited at p. 61]

[Martin, 1996] Martin, K. D. (1996). Automatic transcription of simple polyphonic music: . . . [cited at p. 55]

[Martin, 1982] Martin, P. (1982). Comparison of pitch detection by cepstrum and spectral comb analysis. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82.*, volume 7, pages 180–183. IEEE. [cited at p. 42]

[Martins, 2008] Martins, L. G. (2008). *A computational Framework for Sound Segregation Music Signals*. PhD thesis, University of Porto, Porto, Portugal. [cited at p. 121, 123]

[Meddis and Hewitt, 1991a] Meddis, R. and Hewitt, M. J. (1991a). Virtual pitch and phase sensitivity of a computer model of the auditory periphery. i: Pitch identification. *The Journal of the Acoustical Society of America*, 89:2866. [cited at p. 42]

[Meddis and Hewitt, 1991b] Meddis, R. and Hewitt, M. J. (1991b). Virtual pitch and phase sensitivity of a computer model of the auditory periphery. ii: Phase sensitivity. *The Journal of the Acoustical Society of America*, 89:2883. [cited at p. 42]

[Mendel and Bateson, 1925] Mendel, G. and Bateson, W. (1925). *Experiments in plant-hybridisation /.* Cambridge, Mass. :Harvard University Press,. http://www.biodiversitylibrary.org/bibliography/4532. [cited at p. 63]

[Michalewicz, 1994] Michalewicz, Z. (1994). *Genetic algorithms + data structures = evolution programs (2nd, extended ed.).* Springer-Verlag New York, Inc., New York, NY, USA. [cited at p. 64, 66]

[Miller et al., 1997] Miller, J., Thomson, P., and Fogarty, T. (1997). Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. 219. [cited at p. 70]

[Miller, 1999] Miller, J. F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, GECCO'99, pages 1135–1142, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. [cited at p. 70]

[Miller, 2011] Miller, J. F. (2011). *Cartesian Genetic Programming.* Springer Science & Business Media. [cited at p. XVI, 65, 74, 77]

[Miller, 2013] Miller, J. F. (2013). Gecco 2013 tutorial: cartesian genetic programming. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 715–740. ACM. [cited at p. 92]

[Miller and Harding, 2008] Miller, J. F. and Harding, S. L. (2008). Cartesian genetic programming. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*, pages 2701–2726. [cited at p. 61, 99]

[Miller and Smith, 2006] Miller, J. F. and Smith, S. L. (2006). Redundancy and computational efficiency in cartesian genetic programming. *IEEE Trans. Evolutionary Computation*, 10(2):167–174. [cited at p. 108]

[Miller and Thomson, 2000] Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In *Genetic Programming*, pages 121–132. Springer. [cited at p. 70, 78]

[Miragaia et al., 2021a] Miragaia, R., de Vega, F. F., and Reis, G. (2021a). Evolving a multi-classifier system with cartesian genetic programming for multi-pitch estimation of polyphonic piano music. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 472–480. [cited at p. 132, 138]

[Miragaia et al., 2021b] Miragaia, R., Fernández, F., Reis, G., and Inácio, T. (2021b). Evolving a multi-classifier system for multi-pitch estimation of piano music and beyond: An application of cartesian genetic programming. *Applied Sciences*, 11(7):2902. [cited at p. 132, 138]

[Miragaia et al., 2020] Miragaia, R., Reis, G., de Vega, F. F., and Chávez, F. (2020). Multi pitch estimation of piano music using cartesian genetic programming with spectral harmonic mask. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1800–1807. IEEE. [cited at p. 131, 138]

[Miragaia et al., 2018] Miragaia, R., Reis, G., Fernandéz, F., Inácio, T., and Grilo, C. (2018). Cgp4matlab-a cartesian genetic programming matlab toolbox for audio and image processing. In *International Conference on the Applications of Evolutionary Computation*, pages 455–471. Springer. [cited at p. 88, 93, 131, 138]

[Molla and Torrésani, 2004] Molla, S. and Torrésani, B. (2004). Determining local transientness of audio signals. *Signal Processing Letters, IEEE*, 11(7):625–628. [cited at p. 49]

[Moorer, 1977] Moorer, J. A. (1977). On the transcription of musical sound by computer. *Computer Music journal*, 1(4):32–38. [cited at p. 51]

[Nii, 1986a] Nii, H. P. (1986a). Blackboard systems. *Report No. KSL 86-18 & AI Magazine*, Vols. 7-2 and 7-3. [cited at p. 54]

[Nii, 1986b] Nii, H. P. (1986b). Blackboard systems, part one: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, 7(2):38–53. [cited at p. 54]

[Noll, 1967] Noll, A. (1967). Cepstrum pitch determination. *The journal of the acoustical society of America*, 41(2):293–309. [cited at p. 39]

[Noll and Schroeder, 1964] Noll, A. M. and Schroeder, M. R. (1964). Short-time "cepstrum" pitch detection. *The Journal of the Acoustical Society of America*, 36(5):1030–1030. [cited at p. 120]

[Ochiai et al., 2012] Ochiai, K., Kameoka, H., and Sagayama, S. (2012). Explicit beat structure modeling for non-negative matrix factorization-based multipitch analysis. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 133–136. [cited at p. 58]

[Oliveto and Witt, 2012] Oliveto, P. S. and Witt, C. (2012). On the analysis of the simple genetic algorithm. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 1341–1348, New York, NY, USA. ACM. [cited at p. 64]

[Olson, 1967] Olson, H. F. (1967). *Music, physics and engineering*, volume 1769. Courier Corporation. [cited at p. 89]

[Oppenheim et al., 1997] Oppenheim, A., Willsky, A., and Nawab, S. (1997). *Signals and systems*. Prentice-Hall signal processing series. Prentice Hall. [cited at p. 19]

[Oppenheim and Schafer, 2004] Oppenheim, A. V. and Schafer, R. W. (2004). From frequency to quefrency: A history of the cepstrum. *IEEE signal processing Magazine*, 21(5):95–106. [cited at p. 39]

[Parsons, 1976] Parsons, T. W. (1976). Separation of speech from interfering speech by means of harmonic selection. *The Journal of the Acoustical Society of America*, 60(4):911–918. [cited at p. 46]

[Peeling and Godsill, 2011] Peeling, P. H. and Godsill, S. J. (2011). Multiple pitch estimation using non-homogeneous poisson processes. *IEEE Journal of Selected Topics in Signal Processing*, 5(6):1133–1143. [cited at p. 56]

[Peeters, 2006] Peeters, G. (2006). Music pitch representation by periodicity measures based on combined temporal and spectral representations. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 5, pages V–V. [cited at p. 54]

[Piszczalski and Galler, 1977] Piszczalski, M. and Galler, B. A. (1977). Automatic music transcription. *Computer Music Journal*, 1(4):24–31. [cited at p. 51]

[Poli et al., 2008] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). A field guide to genetic programming. *Published via http://lulu. com and freely available at http://www. gp-field-guide. org. uk.(With contributions by JR Koza)*. [cited at p. XV, 68, 69, 70]

[Rao et al., 2007] Rao, K. S., Prasanna, S. R. M., and Yegnanarayana, B. (2007). Determination of instants of significant excitation in speech using hilbert envelope and group delay function. *IEEE Signal Processing Letters*, 14(10):762–765. [cited at p. 123]

[Rechenberg, 1973] Rechenberg, I. (1973). *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog. [cited at p. 66, 67]

[Reis, 2012] Reis, G. (2012). *Una AproximaciÃ³n GenÃ©tica a la TranscripciÃ³n AutomÃ¡tica de MÃºsica*. PhD thesis, New York, NY, USA. [cited at p. XIV, XV, 18, 20, 36, 41, 42, 48, 61]

[Reis et al., 2012] Reis, G., De Vega, F. F., and Ferreira, A. (2012). Automatic transcription of polyphonic piano music using genetic algorithms, adaptive spectral envelope modeling, and dynamic noise level estimation. *IEEE transactions on audio, speech, and language processing*, 20(8):2313–2328. [cited at p. 59, 95]

[Reis et al., 2007] Reis, G., Fonseca, N., and Ferndandez, F. (2007). Genetic algorithm approach to polyphonic music transcription. In *2007 IEEE International Symposium on Intelligent Signal Processing*, pages 1–6. IEEE. [cited at p. 58]

[Reis and Vega, 2007] Reis, G. and Vega, F. F. (2007). Electronic synthesis using genetic algorithms for automatic music transcription. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1959–1966. [cited at p. 58]

[Röbel, 2003] Röbel, A. (2003). Transient detection and preservation in the phase vocoder. In *Proc. Int. Computer Music Conference (ICMC)*, pages 247–250. [cited at p. 49]

[Rodet and Jaillet, 2001] Rodet, X. and Jaillet, F. (2001). Detection and modeling of fast attack transients. In *Proceedings of the International Computer Music Conference*, pages 30–33. [cited at p. 49]

[Ross et al., 1974] Ross, M., Shaffer, H., Cohen, A., Freudberg, R., and Manley, H. (1974). Average magnitude difference function pitch extractor. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 22(5):353 – 362. [cited at p. 38]

[Rudolph, 1998] Rudolph, G. (1998). Finite markov chain results in evolutionary computation: a tour d'horizon. *Fundam. Inf.*, 35(1-4):67–89. [cited at p. 64]

[Rutkowski, 2008] Rutkowski, L. (2008). Computational. *Intelligence Methods and, Techniques,-: Springer-verlag Berlin Heidelberg.* [cited at p. 67]

[Santoro and Cheng, 2009] Santoro, C. A. and Cheng, C. I. (2009). Multiple f0 estimation in the transform domain. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, pages 165–170, Kobe, Japan. `http://ismir2009.ismir.net/proceedings/PS1-19.pdf`. [cited at p. 53]

[Schroeder, 1968] Schroeder, M. (1968). Period histogram and product spectrum: New methods for fundamental-frequency measurement. *The Journal of the Acoustical Society of America*, 43(4):829–834. [cited at p. 42]

[Schwefel, 1975] Schwefel, H. P. (1975). Evolutionsstrategie und numerische optimierung. *PhD Thesis.* [cited at p. 66]

[Shannon, 1949] Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21. [cited at p. 11, 26]

[Shields, 1970] Shields, V. C. (1970). *Separation of added speech signals by digital comb filtering.* PhD thesis, Massachusetts Institute of Technology. [cited at p. 51]

[Shorten and Khoshgoftaar, 2019] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60. [cited at p. 124]

[Sigtia et al., 2016] Sigtia, S., Benetos, E., and Dixon, S. (2016). An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(5):927–939. [cited at p. 60]

[Smaragdis and Brown, 2003] Smaragdis, P. and Brown, J. (2003). Non-negative matrix factorization for polyphonic music transcription. In *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.*, pages 177–180. [cited at p. 57]

[Smaragdis et al., 2008] Smaragdis, P., Raj, B., and Shashanka, M. (2008). Sparse and shift-invariant feature extraction from non-negative data. *2008 IEEE International Conference on Acoustics Speech and Signal Processing*, pages 2069–2072. [cited at p. 58]

[Su and Yang, 2015] Su, L. and Yang, Y. H. (2015). Combining spectral and temporal representations for multipitch estimation of polyphonic music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(10):1600–1612. [cited at p. 52, 54]

[Tolonen and Karjalainen, 2000] Tolonen, T. and Karjalainen, M. (2000). A computationally efficient multipitch analysis model. *IEEE transactions on speech and audio processing*, 8(6):708–716. [cited at p. 139]

[Vassilev and Miller, 2000] Vassilev, V. K. and Miller, J. F. (2000). The advantages of landscape neutrality in digital circuit evolution. In *Evolvable systems: from biology to hardware*, pages 252–263. Springer. [cited at p. 108]

[Vincent et al., 2010] Vincent, E., Bertin, N., and Badeau, R. (2010). Adaptive harmonic spectral decomposition for multiple pitch estimation. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):528–537. [cited at p. 58]

[Virtanen, 2003] Virtanen, T. (2003). Algorithm for the separation of harmonic sounds with time-frequency smoothness constraint. In *In: Proc. Int. Conf. on Digital Audio Effects*, pages 35–40. [cited at p. 47]

[Walmsley et al., 1998] Walmsley, P. J., Godsill, S. J., and Rayner, P. J. W. (1998). Multidimensional optimisation of harmonic signals. pages 2033–2036. [cited at p. 56]

[Walmsley et al., 1999] Walmsley, P. J., Godsill, S. J., and Rayner, P. J. W. (1999). Polyphonic pitch tracking using joint Bayesian estimation of multiple frame parameters. *Applications of Signal Processing to Audio and Acoustics, 1999 IEEE Workshop on*, pages 119–122. [cited at p. 56]

[Wood, 2007] Wood, A. (2007). *The Physics of Music.* Read Books. [cited at p. 47]

[Yeh, 2008a] Yeh, C. (2008a). *Multiple Fundamental Frequency Estimation of Polyphonic Recordings.* Thése de doctorat, University Paris 6 (UPMC), Paris. [cited at p. XV, 36, 45, 46, 48, 49, 61]

[Yeh, 2008b] Yeh, C. (2008b). Multiple fundamental frequency estimation of polyphonic recordings. *ThÃ©se de doctorat, University Paris 6 (UPMC).* [cited at p. 51, 52]

[Yeh et al., 2006] Yeh, C., Röbel, A., and Rodet, X. (2006). Multiple f0 tracking in solo recordings of monodic instruments. In *120th AES Convention*, pages 20–23. Citeseer. [cited at p. 50]

[Yeh and Roebel, 2006] Yeh, C. and Roebel, A. (2006). Adaptive noise level estimation. In *Workshop on Computer Music and Audio Technology (WOCMAT'06)*, Taipei, Taiwan. [cited at p. 52]

[Yeh and Roebel, 2009] Yeh, C. and Roebel, A. (2009). The expected amplitude of overlapping partials of harmonic sounds. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3169–3172. [cited at p. 47]

[Yeh et al., 2009] Yeh, C., Roebel, A., and Rodet, X. (2009). Multiple fundamental frequency estimation and polyphony inference of polyphonic music signals. *IEEE transactions on audio, speech, and language processing*, 18(6):1116–1126. [cited at p. 90]

[Yeh et al., 2010] Yeh, C., Roebel, A., and Rodet, X. (2010). Multiple fundamental frequency estimation and polyphony inference of polyphonic music signals. *IEEE Transactions on Audio, Speech, and Language Processing*, 18:1116–1126. [cited at p. 43, 52]

[Yoshii and Goto, 2012] Yoshii, K. and Goto, M. (2012). A nonparametric bayesian multipitch analyzer based on infinite latent harmonic allocation. *Trans. Audio, Speech and Lang. Proc.*, 20(3):717–730. [cited at p. 57]

[Zhang et al., 2020] Zhang, W., Chen, Z., and Yin, F. (2020). Multi-pitch estimation of polyphonic music based on pseudo two-dimensional spectrum. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:2095–2108. [cited at p. 55, 139]

[Zhang and Yang, 2015] Zhang, Y. and Yang, Y. (2015). Cross-validation for selecting a model selection procedure. *Journal of Econometrics*, 187(1):95–112. [cited at p. 132]

Appendices

Appendix A

Publications

The work presented in this Thesis is original work undertaken between September 2015 and September 20021 at the University of Extremadura, Spain. Portions of this work have been published elsewhere.

Journal Proceedings

- Miragaia, R., Fernández, F., Reis, G., and Inácio, T. (2021). "Evolving a Multi-Classifier System for Multi-Pitch Estimation of Piano Music and Beyond: An Application of Cartesian Genetic Programming." *Applied Sciences, 11(7), 2902.*
  DOI: `https://doi.org/10.3390/app11072902`

Conference Proceedings

- T. Inácio, R. Miragaia, G. Reis, C. Grilo and F. Fernandéz, "Cartesian genetic programming applied to pitch estimation of piano notes". In *Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1-7. Athens, Greece. IEEE. 2016
  DOI: `https://doi.org/10.1109/SSCI.2016.7850046`

- Miragaia, R., Reis, G., Fernandéz, F., Inácio, T., Grilo, C. "CGP4Matlab-A Cartesian Genetic Programming MATLAB Toolbox for Audio and Image Processing". In *International Conference on the Applications of Evolutionary Computation (EvoApplications).* pp. 455–471. Parma, Italy. Springer: Berlin/Heidelberg, Germany. 2018.
  DOI: `https://doi.org/10.1007/978-3-319-77538-8_31`

- Miragaia, R., Reis, G., de Vega, F.F., Chávez, F. "Multi Pitch Estimation of Piano Music using Cartesian Genetic Programming with Spectral Harmonic Mask". In *Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. pp. 1800–1807, Canberra, Australia, 1–4 December, IEEE, 2020.
  DOI: `https://doi.org/10.1109/SSCI47803.2020.9308178`

- Miragaia, R., de Vega, F. F., and Reis, G. "Evolving a multi-classifier system with cartesian genetic programming for multi-pitch estimation of polyphonic piano music." In *Proceedings of the 36th Annual ACM Symposium on Applied Computing.* pp. 472-480. ACM, 2021.
  DOI: `https://doi.org/10.1145/3412841.3441927`

Appendix B

Symbolic regression Example

Symbolic regression using CGP4Matlab toolbox for the polynomial function:

$$y(x) = x^6 - 2(x^4) + x^2. \tag{B.1}$$

B.1 Configuration

The method for instantiating a new CGP application is by referencing the toolbox **cgptoolbox** and the *CGP* class. Listing B.1 illustrates one possible configuration, and instantiation of the *CGP* Class:

Listing B.1: Example of initializing the CGP Class.

```
% create a configuration struct
configuration = struct(
    'rows', 1,              ...
    'columns', 10,          ...
    'levels_back', 10,      ...
    'output_type', 'random', ...
    'runs', 100,            ...
    'outputs', 1,           ...
    'generations', 8000,    ...
    'offspring', 9,        ...
    'mutation', 0.02,       ...
    'fitness_solution', 0.01, ...
    'fitness_operator', '<=' ...
);

% initialize a CGP instance with a custom configuration
cgp = cgptoolbox.CGP(configuration);
```

B.2 Inputs

Since we are trying to solve a symbolic regression problem, we will create 50 points, between -1 and 1, with Equation 5.1. Those points will constitute the program input (see Listing B.2).

Listing B.2: Example of adding inputs to the CGP.

```
% initialize CGP instance
cgp = cgptoolbox.CGP(configuration);

% create 50 points of x^6 - 2*(x^4) + x^2, between -1 and 1.
a = zeros(1, 50);
b = zeros(1, 50);
index = 1;
for x = -1:2/50:1
    a(index) = x^6 - 2*(x^4) + x^2;
    b(index) = x;
    index = index + 1;
end

% add program inputs
cgp.addInputs( ...
    struct( ...
        'points', struct('x', b, 'y', a) ...
    ) ...
);
```

B.3 Parameters

The following listing shows how to add two parameters to the genotype.

This example will not be used in the symbolic regression problem, because there is no need for additional parameters in the genotype in that case.

Listing B.3: Example of adding two parameteres to the genotype.

```
% initialize CGP instance
cgp = cgptoolbox.CGP(configuration);

% add the parameters to the genotype
cgp.addParameters(
    struct(
        'name', 'some-parameter',
        'initialize', @()rand(),
        'mutate', @(x) x + rand()
    ),
    struct(
        'name', 'constant',
        'initialize', @()randi([-10, 10]),
        'mutate', @mutateParameter
    )
);
```

These parameters will be encoded in the genotype and share the mutation probability with the rest of the genotype's genes. Listing B.4 shows a possible mutation function for a parameter.

Listing B.4: Example of one function that doubles the parameter value at every mutation.

```
function newValue = mutateParameter(parameter)
    % mutate the old parameter value to a new one
    newValue = parameter * 2;
end
```

B.4 Function Set

All the functions from the function-set should be under the same directory. The path to this directory is passed to the CGP's public method *addFunctionsFromPath* (see Listing B.5).

Listing B.5: Example of adding the path to the function-set.

```
% initialize CGP instance
cgp = cgptoolbox.CGP(configuration);

% set the directory of the function-set
cgp.addFunctionsFromPath('./my-path/function-set/')
```

Listing B.6 lists four functions used for the symbolic regression problem. Each function is in a separated file, under *'./my-path/function-set/'*.

Listing B.6: Example of four functions that receive two inputs and do some action with those.

```
% function that sums the first and second input
function result = Sum(x, y)
    result = plus(x, y);
end

% function that subtracts the first and second input
function result = Subtract(x, y)
    result = x - y;
end

% function that divides the first and second input
function result = Divide(x, y)
    if y == 0
        result = x;
    else
        result = x / y;
    end
end

% function that multiplies the first and second input
function result = Times(x, y)
    result = x * y;
end
```

B.5 Fitness Function

The fitness function is provided by callback, which means that we pass the reference of this function to the toolbox. The public method for this is *addFitnessFunction* (see Listing B.7).

Listing B.7: Example of passing the Fitness function to the program.

```
% initialize a CGP instance with a custom configuration
cgp = cgptoolbox.CGP(configuration);

% pass the fitness function as reference
cgp.addFitnessFunction(@myFitnessFunction);
```

The fitness function receives a struct with a series of fields that help with the decodification of the phenotype (see Section ??). The fitness function should return a double or integer for the fitness value. Listing B.8 shows how to decode a phenotype from the symbolic regression problem, step by step. The goal of the symbolic regression fitness function is to minimize the difference between the output of a candidate program and the required output. The fitness (f) is computed by applying the absolute sum of the errors:

$$f = \sum_{t=1}^{50} |e_t|. \tag{B.2}$$

The lower the fitness, the lower the error and the better this candidate solution.

Listing B.8: Example of a fitness function.

```
function score = fitness(args)
    score = 0.0;
    values = zeros(1, args.config.sizes.nodes);

    % for each data point or test case
    for j = 1:50
        values(1) = args.programInputs.points.x(j);

        % iterate through active nodes
        for i = args.config.sizes.inputs + 1:size(args.activeNodes, 2)
            % get current active node that we want to decode
            currentActiveNode = args.activeNodes(i);

            % get the gene that points to the function-gene of the active node
            functionGeneOfActiveNode =
                args.config.structure.functionGenes(currentActiveNode);

            % get the function gene of the active node
            currentFunctionGene = args.genes(functionGeneOfActiveNode);

            % get the genes index
            geneFirstConnection =
                args.config.structure.connectionGenes{1}(currentActiveNode);
            geneSecondConnection =
                args.config.structure.connectionGenes{2}(currentActiveNode);

            % get the nodes index
            nodeFirstConnection = args.genes(geneFirstConnection);
            nodeSecondConnection = args.genes(geneSecondConnection);

            % get the values of the connections
            firstConnection = values(nodeFirstConnection);
            secondConnection = values(nodeSecondConnection);
            % call the function which index is given by currentFunctionGene
            values(currentActiveNode) =
                args.functionSet{currentFunctionGene}(firstConnection,
                secondConnection);
        end

        % compute the sum of squared error
        score = score + abs(values(args.activeNodes(end)) -
            args.programInputs.points.y(j));
    end
end
```

If the genotype is encoded with parameters, those are very easy to extract. Consider the node connection inputs as *ci*, the number of parameters as *np* and the number of genes per node as *g*. If $ci = 2$ and $np = 2$, then $g = 5$: one gene for the function gene, two genes for the node's connection inputs, and two additional genes for the parameters. To recall, the genotype is a sequence of numbers, the genes composing the nodes and the program outputs. Each node starts with the function gene, then the connection inputs and lastly are the parameters. So, if the user wants to extract the parameters relative to node 2, all it needs is to address the last genes from the node. Listing B.9 is the continuation of Listing B.8, extended to decode the parameters from the genotype and pass them to each function call. The functions from the function-set should know which parameters to use and which to ignore.

Listing B.9: Example of decoding the parameters of the current node.

```
for i = args.config.sizes.inputs + 1:size(args.activeNodes, 2)
    % decode the firstConnection and secondConnection
    ...

    % find how many genes per node
    genesPerNode = 3 + args.config.sizes.parameters;

    % genes of the active node
    lastParameter = currentActiveNode * genesPerNode;

    % gene of the first parameter
    firstParameter = lastParameter - args.config.sizes.parameters + 1;

    % get all parameter genes
    allParameters = firstParameter:lastParameter;

    % get the value of each parameter
    for k = 1:size(allParameters, 2)
        parameters(k) = args.genes(allParameters(k));
    end

    % pass the parameters to the functions
    values(currentActiveNode) =
        args.functionSet{currentFunctionGene}(firstConnection, secondConnection,
        parameters);
end
```

B.6 Callbacks

All callbacks receive a struct object with specific properties, relevant to each event.

B.6.1 Generation Ended

After running a generation, the callback *GENERATION_ENDED* is fired. This is very useful for knowing the genes present or the fitness value at each generation. The callback accepts a structure with a few fields, such as the current generation and the fitness value. In Listing B.10, the function will print to the output window the generations and corresponding fitness.

Listing B.10: Example of passing the GENERATION_ENDED callback function to the program.

```
function myGenerationCallback(args)
    % print current generation and fitness
    fprintf('%d - %.16f\n', args.generation, args.fitness);
end

% initialize a CGP instance with some custom configuration
cgp = cgptoolbox.CGP(configuration);

% pass the callback function as reference
cgp.addCallbacks(struct(
        'GENERATION_ENDED', @myGenerationCallback
));
```

B.6.2 Run Ended

At the end of a run, the callback *RUN_ENDED* is fired. This is very useful for knowning the fitness value of each run. The callback accepts a structure with a few fields, such as the current run. In Listing B.11, the function will print to the output window the run that is currently being processed.

Listing B.11: Example of passing the RUN_ENDED callback function to the program.

```
function myRunCallback(args)
    % print current run
    fprintf('run: %d\n', args.run);
end

% initialize a CGP instance with some custom configuration
cgp = cgptoolbox.CGP(configuration);

% pass the callback function as reference
cgp.addCallbacks(struct(
        'RUN_ENDED', @myRunCallback
));
```

B.6.3 New Solution In Generation

When a new solution is generated a *NEW_SOLUTION_IN_GENERATION* event is fired. This is useful to know exactly which offspring is being evaluated at each time. The callback accepts a structure with a few fields, such as the current fitness and current offspring. In Listing B.12, the function will print to the output window the offspring that is currently being processed.

Listing B.12: Example of passing the NEW_SOLUTION_IN_GENERATION callback function to the program.

```
function myNewSolutionCallback(args)
    % print current offspring
    fprintf('current offspring: %d\n', args.offspringIndex);
end

% initialize a CGP instance with some custom configuration
cgp = cgptoolbox.CGP(configuration);

% pass the callback function as reference
cgp.addCallbacks(struct(
        'NEW_SOLUTION_IN_GENERATION', @myNewSolutionCallback
));
```

B.6.4 Genotype Mutated

Everytime a solution is created and the genotype is mutated, a *GENOTYPE_MUTATED* event is fired. This event is useful in order to know which genes were mutated. The callback accepts a structure with three fields: genes before mutation, genes after mutation and index of the mutated genes. Example B.13 shows a function that prints the mutated genes.

Listing B.13: Example of writing to a file the genes before and after mutation.

```
function genotypeMutatedCallback(args)
    % open a file and store in someFileHandler variable
    before = args.genesBeforeMutation(args.genesMutated);
    after = args.genesAfterMutation(args.genesMutated);
    dlmwrite(someFileHandler, [args.genesMutated(:), before(:), after(:)],
        '-append');
end
```

B.6.5 Fittest Solution Found In A Run

When a new solution is better then a previous achieved one, the *FITTEST_SOLUTION* event is fired. The callback accepts a structure with a few fields, such as the current fitness and current run. Example B.14 prints to the output window all the runs that obtained a better candidate solution.

Listing B.14: Example of passing the FITTEST_GENERATION callback function to the program.

```
function myFittestSolutionCallback(args)
    % print current generation and fitness
    fprintf('new fittest solution at run: %d \n', args.run);
end

% initialize a CGP instance with some custom configuration
cgp = cgptoolbox.CGP(configuration);

% pass the callback function as reference
cgp.addCallbacks(struct(
        'FITTEST_SOLUTION', @myFittestSolutionCallback
));
```

B.6.6 Fittest Solution Of A Generation

When iterating through the offspring, if a solution is better than the previous one, the event *FITTEST_SOLUTION_OF_GENERATION* is fired. The callback accepts a structure with a few fields, such as the current fitness and current offspring. Example B.15 prints to the output window the new fittest offspring index.

Listing B.15: Example of passing the FITTEST_SOLUTION_OF_GENERATION callback function to the program.

```
function myFittestSolutionOfGenerationCallback(args)
   fprintf('new fittest offspring: %d \n', args.offspringIndex);
end

% initialize a CGP instance with some custom configuration
cgp = cgptoolbox.CGP(configuration);

% pass the callback function as reference
cgp.addCallbacks(struct(
       'FITTEST_SOLUTION_OF_GENERATION', @myFittestSolutionOfGenerationCallback
));
```

Appendix C

Test results for first approach (CGP-1)

All the classifiers were tested after the evolutionary process with chords and isolate notes not previously used in the training stage. Table C.1 shows the detailed results for 61 classifiers. From classifier pitch number 36 to 96. Evaluation process using F-measure for the test cases: **tp** True positives, **tn** True negatives, **fp** False positives, **fn** False negatives.

Table C.1: Test results for (CGP-1) of 61 classifiers

classifier	tp	tn	fp	fn	F-measure
36	5	140	4	0	0.714
37	5	121	23	0	0.303
38	5	140	4	0	0.714
39	5	113	31	0	0.244
40	4	130	14	1	0.348
41	4	138	6	1	0.533
42	4	124	20	1	0.276
43	4	138	6	1	0.533
44	5	112	32	0	0.238
45	5	135	9	0	0.526
46	3	138	6	2	0.429
47	4	119	25	1	0.235
48	5	135	9	0	0.526
49	5	136	8	0	0.556
50	5	140	4	0	0.714
51	5	127	17	0	0.370
52	5	138	6	0	0.625
53	5	141	3	0	0.769
54	5	128	16	0	0.385
55	5	138	6	0	0.625
56	5	128	16	0	0.385
57	5	139	5	0	0.667
58	5	139	5	0	0.667
59	5	137	7	0	0.588
60	5	142	2	0	0.833
61	4	142	2	1	0.727
62	4	143	1	1	0.800
63	4	144	0	1	0.889
64	5	138	6	0	0.625
65	5	141	3	0	0.769
66	5	139	5	0	0.667
67	5	141	3	0	0.769
68	5	140	4	0	0.714
69	5	141	3	0	0.769
70	5	142	2	0	0.833
71	5	141	3	0	0.769
72	5	142	2	0	0.833
73	5	140	3	0	0.769
74	4	142	2	1	0.727
75	4	142	2	1	0.727
76	5	139	5	0	0.667
77	4	142	2	1	0.727
78	4	140	4	1	0.615
79	5	140	4	0	0.714
80	4	140	4	1	0.615
81	5	143	1	0	0.909
82	5	144	0	0	1.000
83	5	142	2	0	0.833
84	5	145	2	0	0.833
85	5	144	3	0	0.769
86	5	146	1	0	0.909
87	5	141	3	0	0.769
88	5	142	2	0	0.833
89	5	142	2	0	0.833
90	5	143	1	0	0.909
91	4	144	0	1	0.889
92	5	143	1	0	0.909
93	5	140	3	0	0.769
94	5	141	3	0	0.769
95	5	143	2	0	0.833
96	5	141	3	0	0.769

Appendix D

Extended Function Set

List of all the 27 functions that make up the **Function-set**, trough a look up table. function indices starts in 1 to 27. There are function with arity 1, use only one input argument, and functions that use 2 input arguments, arity 2.

Table D.1: Function set lookup table

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

Appendix E

Configuration file for k-fold cross validation

Listing E.1: k-fold configuration config.txt

```
[signal]
sampling_frequency=44100
samples=4096
threshold=1.5
type=record
triangular_signal=0 0.5 1 0.5 0
triangular_signal_2=0 0.2 0.3 0.2 0
function_set=transcription/function-set
fitness_function=fmeasure
interval_standard_deviation=0.3
constant_standard_deviation=3
fft_samples=half
second_harmonic=1
test_cases=500
threshold_increment=0.01
polyphony=chords
inputs=real,imag,abs,angle,cceps


[cgp]
rows=1
columns=100
levels_back=100
output_type=last
outputs=1
generations=10000
runs=10
```

```
offspring=4
mutation=0.05
threshold_mutation=0.1
fitness_solution=1
fitness_operator=>=

[report]
generation_frequency=500
callbacks=CONFIGURATION,RUN_ENDED,FITTEST_SOLUTION,GENERATION_ENDED,
NEW_SOLUTION_IN_GENERATION,FITTEST_SOLUTION_OF_GENERATION,FIGURES
```

Appendix F

Classifier 55 decoded example

Listing F.1: node genotype decoded

```
generation:10000
fitness:0.9523809523809523
threshold:1.2874124589452547
true positives:370
true negatives:367
false positives:20
false negatives:17

Node Index1 = input (1)
Node Index2 = input (2)
Node Index3 = input (3)
Node Index4 = input (4)
Node Index5 = input (5)
 Computational Node 1 = SPLog ( 2 , 1 )
 Computational Node 2 = SPSumConst ( 3 , 1 )
 Computational Node 5 = SPSin ( 3 , 2 )
 Computational Node 6 = SPTimes ( 5 , 5 )
 Computational Node 8 = SPPeaks ( 4 , 7 )
 Computational Node 11 = SPCos ( 4 , 1 )
 Computational Node 12 = SPCceps ( 5 , 6 )
 Computational Node 14 = SPConvolution ( 7 , 11 )
 Computational Node 18 = SPNormalizeSum ( 16 , 13 )
 Computational Node 19 = SPSin ( 10 , 1 )
 Computational Node 20 = SPSumConst ( 3 , 13 )
 Computational Node 46 = SPTimes ( 17 , 13 )
 Computational Node 49 = SPNormalizeMax ( 23 , 25 )
 Computational Node 67 = SPSumConst ( 54 , 51 )
 Computational Node 90 = SPGaussfilter ( 72 , 19 )
 Computational Node 100 = SPSum ( 95 , 24 )
```

```
active: 1.000000, 2.000000, 3.000000, 4.000000, 5.000000, 6.000000, 7.000000,
    10.000000, 11.000000, 13.000000, 16.000000, 17.000000, 19.000000,
    23.000000, 24.000000, 25.000000, 51.000000, 54.000000, 72.000000,
    95.000000, 105.000000,
```