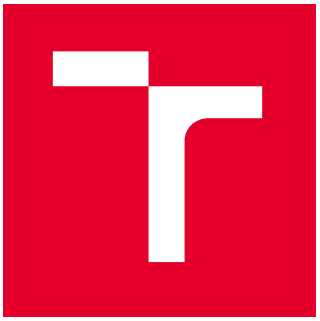


BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING

FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ÚSTAV AUTOMATIZACE A INFORMATIKY

MUSIC COMPOSITION USING ARTIFICIAL INTELLIGENCE METHODS

HUDEBNÍ KOMPOZICE S VYUŽITÍM METOD UMĚLÉ INTELIGENCE

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Katia Vendrame**

AUTOR PRÁCE

SUPERVISOR — **prof. Ing. Radomil Matoušek, Ph.D.**

VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

Assignment Master's Thesis

Institut:	Institute of Automation and Computer Science
Student:	**Bc. Katia Vendrame**
Degree programm:	Applied Computer Science and Control
Branch:	no specialisation
Supervisor:	**prof. Ing. Radomil Matoušek, Ph.D.**
Academic year:	2023/24

As provided for by the Act No. 111/98 Coll. on higher education institutions and the BUT Study and Examination Regulations, the director of the Institute hereby assigns the following topic of Master's Thesis:

Music composition using artificial intelligence methods

Brief Description:

It will be a very complex and interdisciplinary work requiring training in AI (artificial intelligence) and computer science as well as in musicology. The aim will be to build a program for music generation and composition, i.e. to algorithmise composition using AI. For the breadth of the topic, we will focus on Mazurka–style composition by Chopin.

Master's Thesis goals:

– Conduct research on approaches to automatic music generation using evolutionary computational techniques (ECT).
– Conduct research on approaches to automatic music generation using deep neural networks.
– Algorithmatize the composition and build a program (ECT), which will generate music in the style of Chopin's mazurkas.
– Perform a comparative statistical and non–additive sensory analysis of the result of the work.

Recommended bibliography:

Ian Simon and Sageev Oore. (2022). Performance rnn: Generating music with expressive timing and dynamics. https://magenta.tensorflow.org/performance-rnn, 2017. (Accessed: 2022-04-11).

Özcan, E., Erçal, T. (2008). A Genetic Algorithm for Generating Improvised Music. In: Monmarché, N., Talbi, EG., Collet, P., Schoenauer, M., Lutton, E. (eds) Artificial Evolution. EA 2007. Lecture Notes in Computer Science, vol 4926. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-79305-2_23

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2023/24

In Brno,

L. S.

Ing. Pavel Heriban, Ph.D. Director of the Institute	doc. Ing. Jiří Hlinka, Ph.D. FME dean

ABSTRACT

This research delves into methods for automatic music composition, with a specific emphasis on evolutionary algorithms and neural networks. It examines the potential correlation and discourse between musicology theories and automatic music composition, as well as its foundation in musical tradition. The study focuses on three algorithms utilized for generating short monophonic melodies stylistically based on given datasets or the user's requirments: probabilistic grammar evolution, genetic algorithms, and LSTM models. The practical part of this work showcases the application of these algorithms and presents results from testing their efficacy and capabilities. Furthermore, it introduces an implementation for analyzing MIDI datasets from a musical perspective. Ultimately the study highlights the potential for future enhancements and broader applications in the field of automatic music analysis and composition.

ABSTRAKT

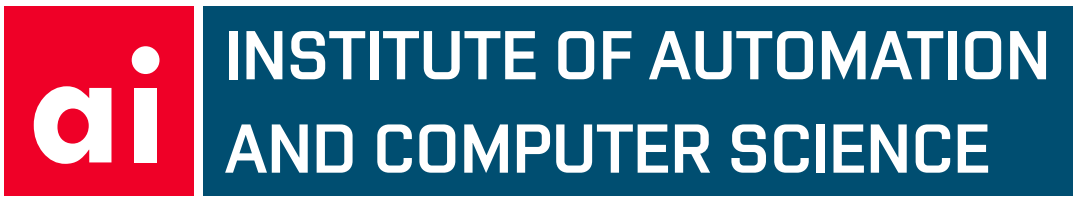
Tato studie se věnuje metodám automatické kompozice hudby (AMC), s konkrétním zaměřením na evoluční algoritmy a neuronové sítě. Potenciální dialog mezi muzikologickými teoriemi a AMC jsou analyzovány, spolu s otázkou jejího základu v hudební tradici. Byly zkoumány tři algoritmy pro tvoření krátkých jednohlasých melodií založených na stylu daného datasetu nebo požadavcích uživatele: pravděpodobnostní gramatická evoluce, genetické algoritmy a LSTM modely. Praktická část práce představuje aplikace těchto algoritmů a výsledky testování jejich výhod a předností. Dále je představena implementace pro analýzu MIDI datasetů z hudební perspektivy. V poslední řadě jsou představeny možnosti budoucího vylepšení a rozšíření zkoumaných algoritmů v oblasti automatické hudební analýzy a kompozice.

KEYWORDS

Automatic music composition, probabilistic grammar evolution, automatic music analysis, genetic algorithm, LSTM neural networks

KLÍČOVÁ SLOVA

Automatická hudební kompozice, pravděpodobnostní gramatická evoluce, automatická hudební analýza, genetický algoritmus, LSTM neuronové sítě



2024

BIBLIOGRAPHIC CITATION

VENDRAME, Katia. *Music composition using artificial intelligence methods*. Brno, 2024. Available at: `https://www.vut.cz/studenti/zav-prace/detail/157901`. Master's Thesis. Brno University of Technology, Faculty of Mechanical Engineering, Institute of Automation and Computer Science, Supervised by prof. Ing. Radomil MatouÅąek, Ph.D.

AUTHOR'S DECLARATION

I declare that this master's thesis is my original work, and I have worked on it independently under the supervision of my master's thesis advisor, using specialized literature and other information sources, all of which are cited in the thesis and listed in the bibliography.

As the author of the stated work, I further declare that in connection with the creation of this work, I have not violated the copyrights of third parties, particularly I have not unlawfully infringed upon the personal copyrights of others, and I am fully aware of the consequences of violating the provisions of Âğ11 and following of the Copyright Act No. 121/2000 Coll., including possible criminal consequences.

In Brno on May 24, 2024 .....

Katia Vendrame

ACKNOWLEDGEMENT

I would like to express my gratitude to my family, who gave me the special opportunity to study and supported me all these years. A special thanks goes to my two amazing sisters and Vít that always stood by me and encouraged me to do better.

CONTENTS

1 PREFACE

Rooted in the Latin verb “componere”, which signifies “putting together” or “unifying”, composition in music embodies the act of weaving disparate sounds into a narrative that resonates with the human experience. Just as a skilled storyteller crafts a narrative by stringing together words, composers manipulate melodies, harmonies, and rhythms to find the right place for each one of them in order to express their stories in the most accurate way possible.

At the heart of musical composition is the concept of juxtaposition - the deliberate arrangement of musical elements to construct a compelling story without words. Each chord, each note, carries its own significance, but it is the relationship between them that gives music a voice.

According to D’Indy’s definition, music composition involves the “arrangement of various elements”, with the art lying in the skillful arrangement of melodies, harmonies, and modulations to give them a unified meaning and form.

In this study, we are exploring various methods of juxtaposing musical elements to replicate the relationships and structures found in existing melodies. Our objective is twofold: to develop a tool to aid the compositional process and to analyze the underlying structures of melodies across different musical styles and composers, discerning the characteristics that distinguish one from another.

The thesis begins by introducing and analyzing various generative evolutionary algorithms applicable to music composition. Following this, the second section delves into the widespread use of neural networks for artistic creation. The third section showcases the implementation of a probabilistic grammatical evolution algorithm, genetic algorithm and LSTM neural network, accompanied by illustrative examples of its outcomes.

2 STATE OF THE ART

The process of analyzing data in the form of musical melodies to produce new music is referred to as automatic music composition (AMC) (Wiafe, 2022). Historically there had been many algorithms that have been explored to reach this goal: stochastic processes, Markov chains, grammars, evolutionary algorithms and neural networks are just a few examples of the methods that have been implemented (Wiafe, 2022). The choice reflects the way music is interpreted.

If the stochastic character of musical pieces is the fundamental feature desired in the outputs, mathematical algorithms for automatic music composition have been applied. The mathematical models most explored with this purpose are the Markov chains. Thanks to their low complexity they have been successfully used in real-time applications (completing melodies, generating music for games) (Liu, 2017), or to create beats for electronic music (Eigenfeldt, 2014).

When the aim is to generate music that has internal structures and rules governing its notes, linguistic models have been applied. Grammars that comprise rules about rhythm and harmony are usually implemented in this case and the generated melodies refer to a specific genre or style, according to the grammar rules used. An example is the Granroth-Wilding's work (Granroth, 2014), where music (described as a language) is generated by an algorithm that learns patterns (jazz chords) from the input dataset and uses them to create new ones (Liu, 2017).

The third main field of application of automatic musical composition are the artificial intelligence methods, and specifically evolutionary computation, machine learning, cellular automata, knowledge-based systems and, mostly in the last years, neural networks.

The first music ever composed by a computer is the ILLIAC suite, described by Hiller and Isaacson in 1958 (De Prisco, 2019). Hiller considers the process of composing music as a process of selection from an infinite universe of sonic material, in other words a way to "extract" order from this universe, using mathematical operations, probability operations and general principles of analysis. To Hiller, this process seamed analogical to the functioning of computers, that create random datasets and then extract material from them, choosing from an infinite variety of possibilities. Therefore, he uses a pseudo-random integer generator, which represents notes and rhythms and selects a set of generated numbers according to some input rules (a process defined as the "Monte-Carlo Method") (Both, 1995). In Fig.1 the scheme of the composition process of Illiac Suite is represented:

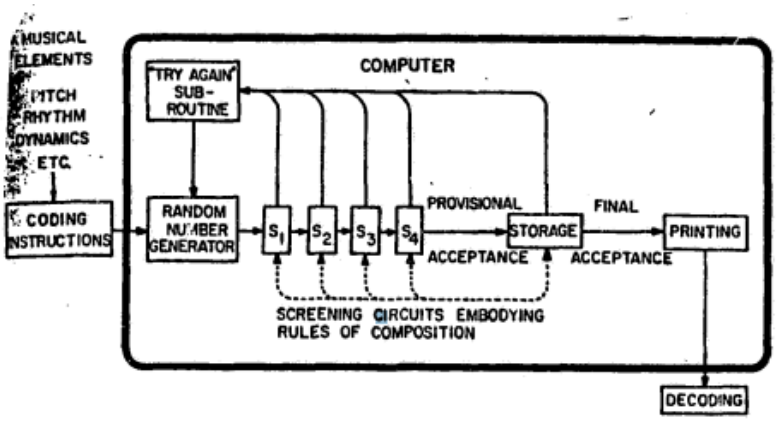


Fig. 1: Block diagram of the composition algorithm for Illiac Suite (Hiller, 1979)

2.1 Evolutionary computing

During the 1980s, Evolutionary Algorithms began to receive significant attention as optimization tools, thanks to their flexibility, adaptability to various tasks, and advancements in computer performance (Back, 1997). It is no wonder, then, that these algorithms were also tested in automatic music composition processes, serving as a tool to explore large solution spaces and locate multiple optimal solutions (Prisco, 2022). These algorithms ensure a continual combination of exploration and optimization, driven by the genetic operators they employ (Goldberg, 2002).

Genetic algorithms were used to generate complete compositions (Jacob, 1995), or any kinds of musical subtasks: melody composition, counterpoint, sound generation, thematic transformation (Miranda, 2007), jazz improvisation, harmonization and so on. An example is the composition system "Variations" designed by Jacob (see Fig.2), that creates phrases combining motives taken from a dataset and then uses variation mechanisms to search the space.

Melody development is the musical task that was most explored with the use of evolutionary algorithms. Systems that generate pitch and rhythm sequences were implemented with the use of different evolutionary algorithms since the 90s (Biles, 1997), (Jacob, 1996) until nowadays (the automatic melody composition presented by (Jeong, 2017) or the MetaCompose application by (Scirea, 2017).

Automatic harmonization was also widely studied since the 90s (Cope, 1997, Miranda, 2000). Four-parts compositions (for example the widely known Bach's 4-voices chorales) address this issue thoroughly. The "EMI system" presented in (Cope, 1997) uses grammars and rules to execute this. In (Gang, 1996) neural networks are used. In (Prisco, 2022) the EvoComposer application is introduced, which explores the potential of evolutionary algorithms in this AMC discipline.

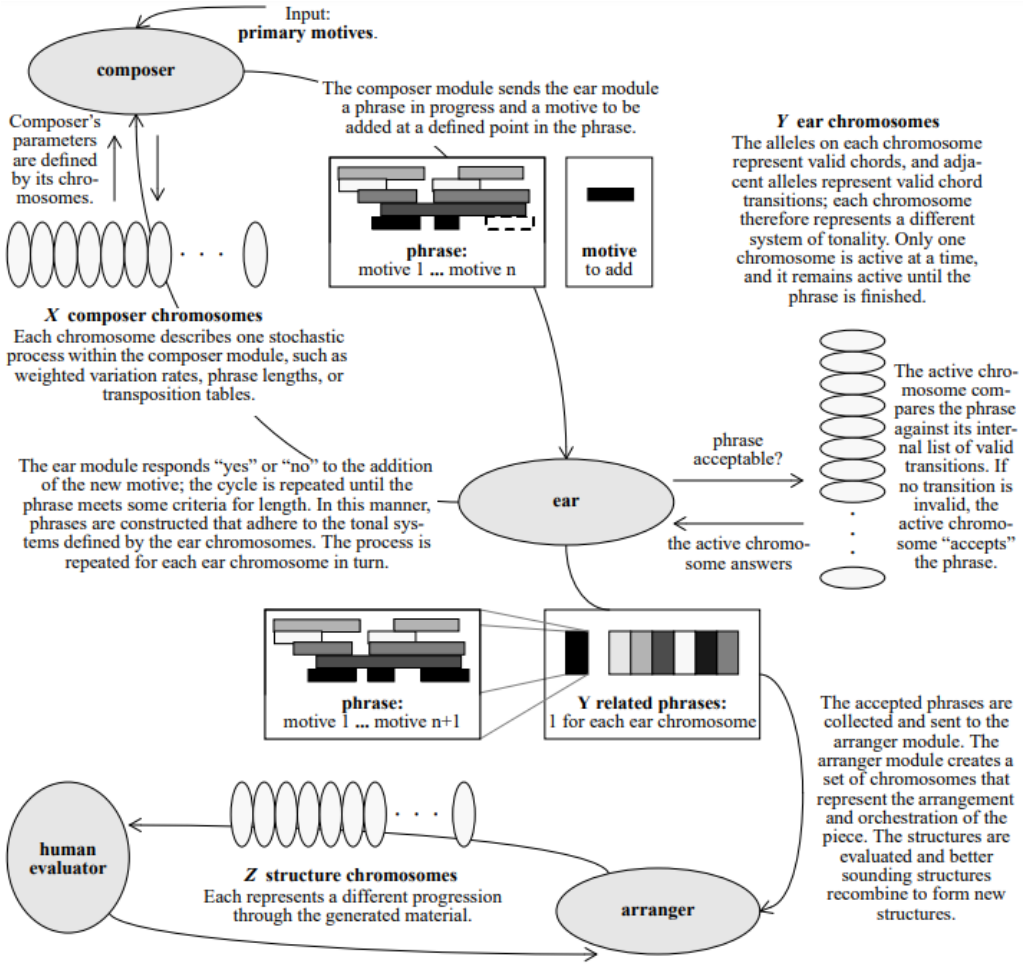


Fig. 2: Block diagram of the composition system "variations" (Jacob, 1995)

The Evocomposer application uses a multiobjective evolutionary algorithm to compose a 4-voices piece according to one voice input (it can be any of the 4 parts). This algorithm implementation is based on the results of a statistical analysis of a dataset (in this case Bach's corales) to derive a table of weights for chords and tonality changes.

Alongside Evolutionary algorithms, grammars were often implemented for music creation as an easy tool to apply strict rules to the free composition process made by a genetic implementation. To make non-deterministic, creative output, grammars can be used in collaboration with evolutionary algorithms, creating Grammatical Evolution implementations. One example is the application GenerativeGI (Fredericks, 2023) developed to create visual art.

2.2 Neural Networks

Thanks to the rapid development of neural networks in the last decade, numerous music applications have experimented with these algorithms. Neural networks are now used for genre and style classification, music creation, music interpretation (performance), and many other applications. The study of artificial neural networks gained significant attention by the 1960s, with the approach of using mathematical, artificial models to study human cognition being termed "connectionism" (Waskan, 2022). One of the main goals of connectionism was to develop computer behavior that closely mimicked human behavior. This field of research was referred to by the Japanese as âĂIJKansei,âĂİ meaning âĂIJenabling computers to express emotionsâĂİ (Bresin, 1998). This focus on emotional expression is why music performances have attracted researchers' attention. In Stockholm, a symbolic rule system was developed to convert music scores into performances that were as emotionally expressive as possible (Friberg, 1991). These results were subsequently used to design an Artificial Neural Network (ANN) (Bresin, 1998).

2.2.1 Historical overview

In "Music and Connectionism", Todd and Loy (1991) explain how connectionism can save AMC from the trap of being strictly formal (Garton, 1995). Isaacson presented neural networks as a tool to study post-tonal music: the music, that can be theorized by the pitch-class set theory: using neural networks he studies the relationships between sets (interval classes, similarities) and how listeners perceive them (Isaacson, 1997).

However, music classification is up until now the most explored musical application of neural networks. For the classification tasks spectrograms are mostly used as inputs: Despois used CNN to classify music into main genre classes as Electro,

Classical, Rap, etc and (Pelchat, 2020) continued the (Despois, 2018) work using CNN and music spectrograms. Alongside spectrograms, also methods taken from speech recognition were used for music recognition tasks (Zhong, 2011).

Music generation using neural networks hasn't been left behind too. In 2015 Marquetti presented a composition inspired by Mozart's dice game, that uses supervised neural networks to compose melodies and improvise in real time (Fig. 3).

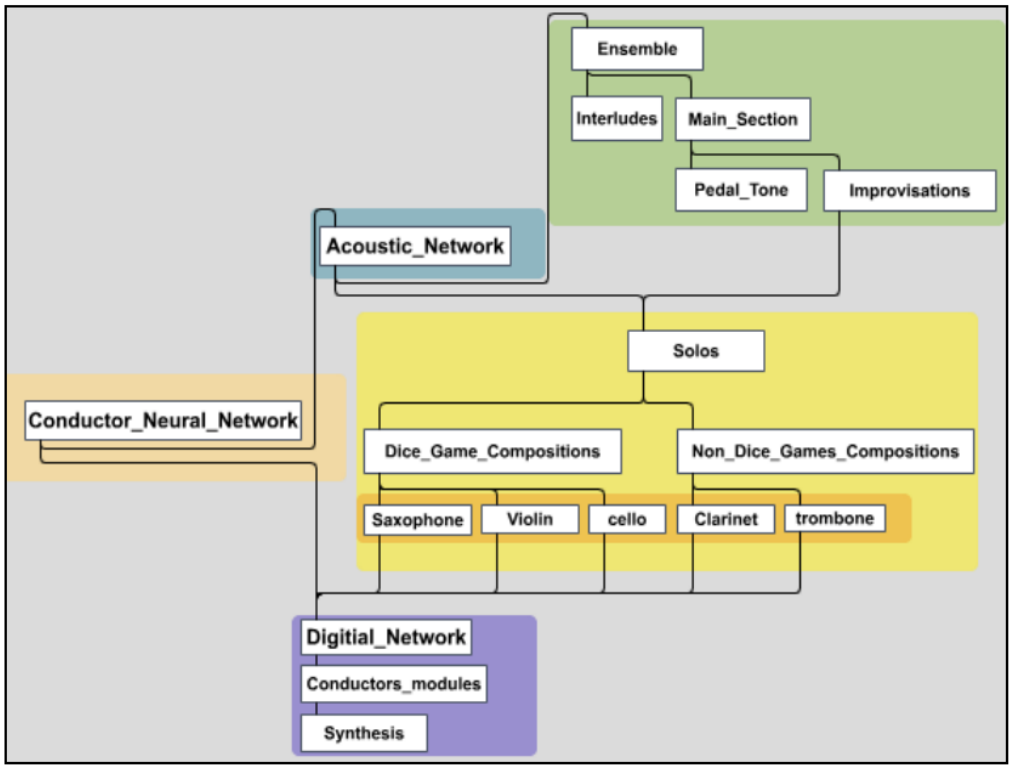


Fig. 3: Block diagram of the composition system "Solos (Dice Game) and Conductor (Neural Network)" (Marquetti, 2015)

In addition to these most explored fields, neural networks are applied to transcript audio recordings to musical scores (Schlüter, 2014), to remix voice and sound balance in musical recordings (Simpson, 2015), musical therapy applications, music recommendation algorithms, music education and others.

Nowadays more and more neural networks trained to compose music are developed. One of the most popular one is the Magenta project: an open-source research project by Google (tensorflow), first released in 2019 and then in 2023 (Magenta Studio 2). The Magenta research group studies deep learning applied to signal processing, music creation, audio synthesis, transcriptions and others. For music creation they research and employ different models: Long-Term Structures (LTS) in the "Music Transformer" application, a hierarchical latent vector model for learning

LTS in the "Music-VAE" application, generation latent constraints, LSTM-based recurrent neural networks and autoencoder models (WaveNet) (Google, 2024).

Music-VAE addresses the difficulty of modelling sequences with long-term structure by proposing the use of a hierarchical decoder. This solution shows a better sampling, interpolation, and reconstruction performance (Roberts, 2018). The variational autoencoder encodes an entire sequence to a single latent vector,

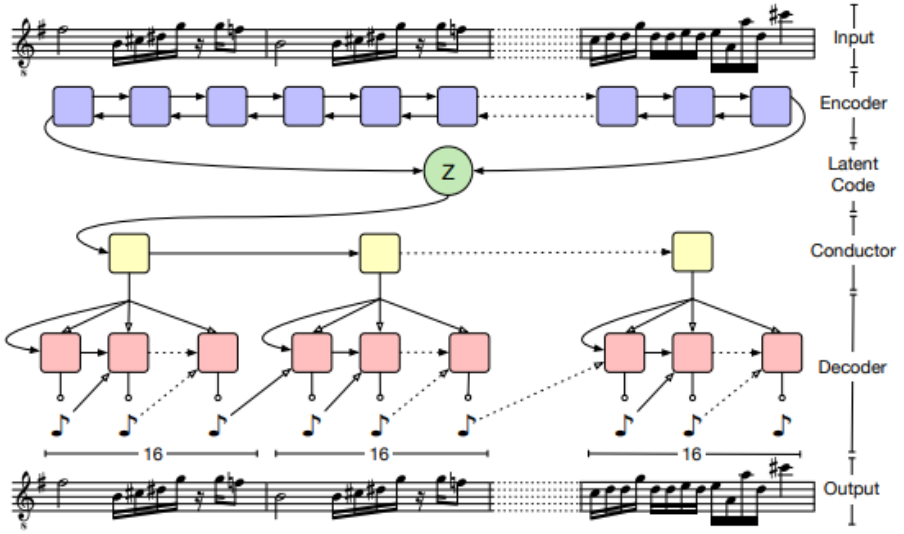


Fig. 4: Architecture scheme of MusicVae (Roberts, 2018)

creating a long-term structure for the generated piece of music, which is almost always present in western popular music. The structure of this implementation can be seen in Fig.4

Another research project in this field is MuseNet by OpenAI: it is a deep neural network, that can generate short musical composition with different instruments and in different styles (from classical to popular music), using a transformer model. As a training set it used MIDI files from a wide range of musical styles (OpenAI, 2019).

One of the most recent open-source applications model is V3 by Suno.ai, which is capable to create radio quality songs from words. It is based on a text-to speech model called Bark, released in 2023 (Freyberg, 2024). Bark compresses and reconstructs audio using EnCodec (Leung, 2023), whose structure is represented in Fig.5.

Alongside the encoding model, Bark uses a GPT-like text (semantic) model and an acoustics model that has the same structure as the text model.

According to Sarmento in (Sarmento, 2023), automatic music generation that uses symbols (instead of waves, e.g. MIDI files) can be categorized according to the

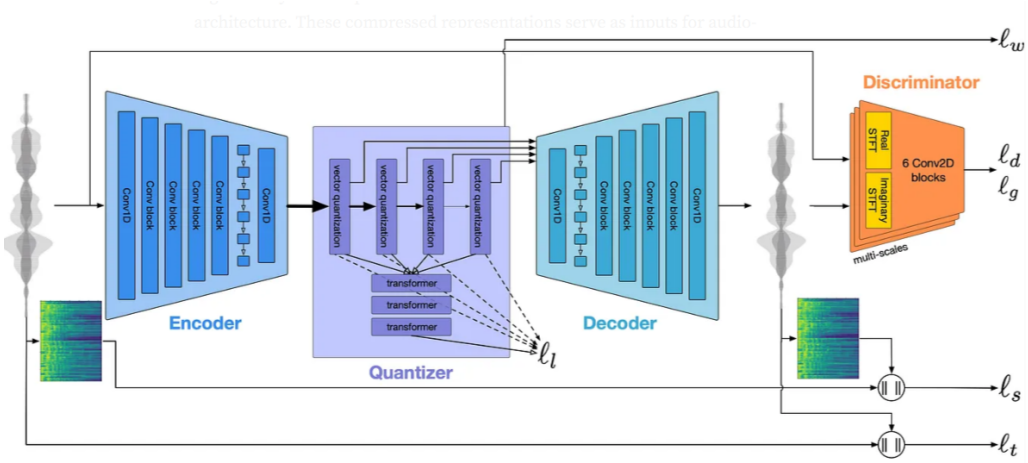


Fig. 5: Architecture scheme of EnCodec (Leung, 2023)

network architecture into three main categories: Variational Autoencoder models (VAEs), Generative Adversarial Networks (GANs), and models that are inspired on natural language processing as Long Short-Term Memory netwokrs (LSTM), Transformers or Recurrent Neural Networks (RNN).

2.2.2 Variational autoencoders

Variational autoencoders are used to generate music according to some rules (structural or stylistic). In (Lim, 2020) for example, it was used to generate music in Bach's style. VAE models have an encoder-decoder architecture and are based on the idea of learning a posterior distribution that approximates the true posterior. It generates new data by learning a continous latent space of music symbols (Vechmotova, 2023). VAE models have been used also for audio composition - in this case the latent space is made of audio frames: they can encode an existing audio frame to a latent space and synthetize frames by interpolation and extrapolation of timbres (Tatar, 2021), generating audios of any length.

2.2.3 Generative adversarial networks

GAN models, on the other hand, are made of a generator and a discriminator, that generate together new data according to some requested characteristics. The generator creates adversarial samples to train the discriminator, that has to learn to discriminate between real and generated melodies. They can be implemented with VAE. For example, in (Vechmotova, 2023), they are used to create lyric lines to a live music performance (Fig.6). A semi-recurrent cnn-based VAE-GAN network was implemented also in (Akbari, 2018) to generate sequences of individual frames generated using CNN. Dong in 2018 in his project MuseGAN, proposes a multi-track

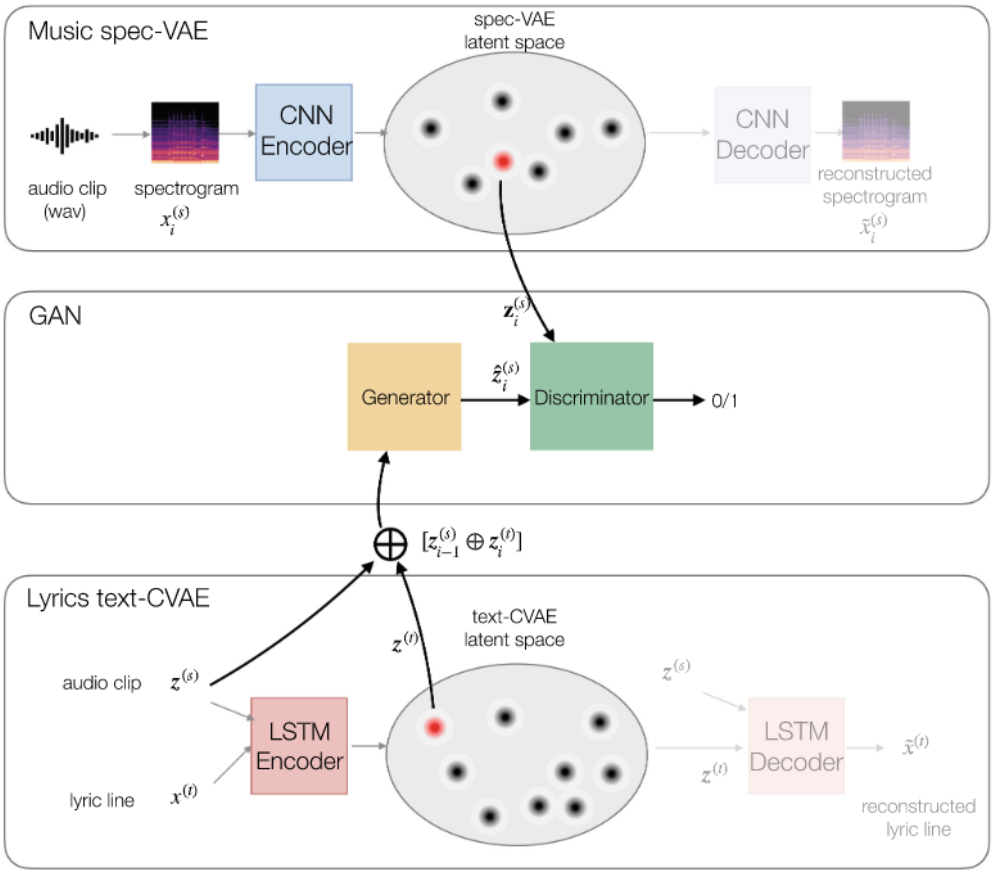


Fig. 6: Architecture scheme of LyricJam Sonic (Vechmotova, 2023)

sequential generative adversarial network for music and accompaniment generation, which has a temporal, harmonic and rhythmic structure. Their network structure is made of two models, inspired to the two main different ways to approach music composition: jamming or composing. The generator of the GAN concatenates random vectors from the shared temporal structure, from the private one (both time dependent) and each of them outputs the inner track and the inter track temporal information (Dong, 2017).

2.2.4 Transformers

The Transformers are sequence-to-sequence models, they are used to create longer sequences of symbolic music, thanks to the self-attention they use to bias the prediction of the current token based on the previous ones (Huang, 2020). One example is Musenet, that uses the GPT-2 model. Furthermore, Music Transformer can compose expressive piano 1-minute pieces, or the Pop Music Transformer (Huang, 2020) uses a variant (Transformer-XL model) of this algorithm to create music from different genres (Sarmento, 2023) of longer length. In addition to longer melody outputs, the Pop Music Transformer proposes an alternative to the MIDI event representation: the REMI representation, which is beat-based - to each note there is the information of which position it has in the measure and what its rhythm is (Huang, 2020).

Transformers are also used to obtain more controlled music generation, e.g. the GTR-CTRL model is a model that generates tab music, controlling the genre and instrumentation (Sarmento, 2023). These models are presented as a solution for one of the main limits of the application of neural networks, which is the difficulty in controlling and conditioning the generative process. The controlling process is realized by control tokens, that are inserted at the beginning of every element of the training dataset.

2.2.5 Long Short-Term Memory networks

The Long Short-Term memory networks are used in many different models. In (Manzelli, 2018) they are used to learn the melodic structure of different music styles and generate symbolic music pieces, which are then used as input for a WaveNet-based audio generator (in order for the output to sound realistic).

Cong Jin et al. in 2020 proposed a style-specific music composition neural network, where as a generator there is a LSTM network, as discriminator a CNN, and an Actor-Critic (AC) network is used to make fine-tuning (Jin, 2020). The LSTM network, a variant of RNN, uses a special timing memory function thanks to its inner gates: its architecture contains three gate structures - output gates, forget gates and input gates. Thanks to its structure it can learn long-term dependencies in the data (Lattner, 2020), an essential ability to create long musical pieces, which

need a long-term structure. On the other hand, RNNs are ring-type networks, where the output of the neural unit depends on the actual input time and to the value at the previous time.

Based on these network typologies (Wang, 2021) proposed a Music Composition Neural Network (MCNN), which uses LSTM to generate music, a reward function that is based on the "basic criteria of music creation", that can adjust the music composition real-time and a CNN as discriminator.

2.2.6 Application examples

One of the major challenges in automatic music composition since its birth, is the composition of polyphonic music pieces in the style of J. S. Bach's chorales. They are used because of their strict harmonic structure, their lengths and mainly because they form a homogenous dataset of an acceptable size (Bach composed 389 chorales) (Hadjeres, 2016). To solve this challenge many different approaches have been used: genetic algorithms , Hidden Markov Models and neural networks (RNN, LSTM, Gated Reccurent Units, Restricted Boltzmann Machines, and so on) (Hadjeres, 2016).

DeepBach is a solution presented by (Hadjeres, 2016): a dependency network capable of producing choreales in the style of Bach, create coherent phrases and harmonize melodies (Fig.7). DeepBach uses four neural networks to take in account

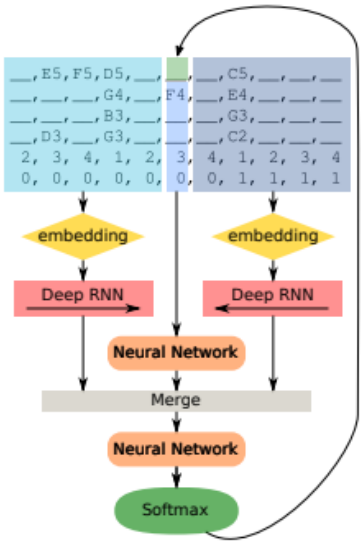


Fig. 7: Architecture scheme of DeepBach (Hadjeres, 2016)

the sequential aspect of the data. One network retrieves information from the past, the other one from the future, then there is a non-recurrent neural network to control notes occurring at the same time. The outputs of these three networks

are the passed to the fourth RNN. This architecture was selected reflecting the compositional practice of these chorales (harmonization is often made starting from the end of the piece) (Hadjeres, 2016).

Based on this overview about how neural networks are applied to music composition can be concluded that the researches are generally concerned mainly on the generation of short music pieces: this means, that until now only very little attention to the higher-level musical characteristics has been paid. This is one of the goals of the work of (Lattner, 2019), in order to generate longer musical compositions. To do that, they created an algorithm, called Constrained sampling that uses a Convolutional Restricted Boltzmann Machine, in combination with a musical dataset for training and structural constraints (the gradient descent method). These cost functions guide the creation of the tonal, metrical, and self-similarity structure of the pieces (Lattner, 2019). The sampling content, which is then modified according to

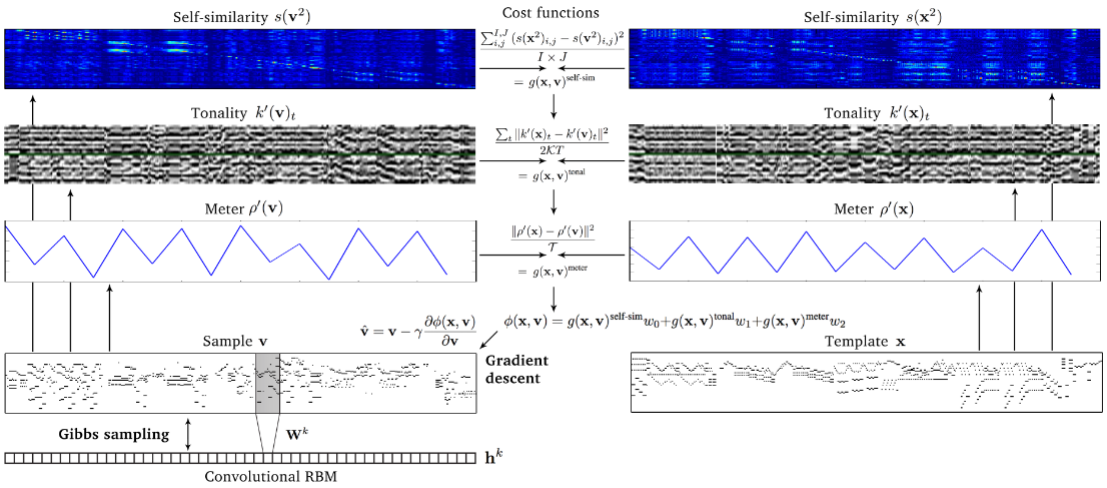


Fig. 8: Architecture scheme of Constrained Sampling Algorithm (Lattner, 2019). Constrained sampling using an existing piece $\mathbf{x}$ as structure template and $\mathbf{v}$ is a randomly initialized sample.

the constraint functions, is built by a two-layered CNN. The visible layer is made of a piano roll representation, as can be seen in Fig.8. The randomly initialized sample is updated with Gibbs sampling (GS) and gradient descent (GD), which lowers the error $\Phi(x, v)$ between the template and the sample. Gibbs sampling runs until the free energy function of the RBM stabilizes. The sample at each iteration is subject to a gradient descent (GD) optimization, which updates the weights according to a learning rate. The cost functions that the GS uses are (Lattner, 2019):

1. the self-similarity constraint, which builds a self-similarity matrix where repeating patterns are highlited (see Fig.9).

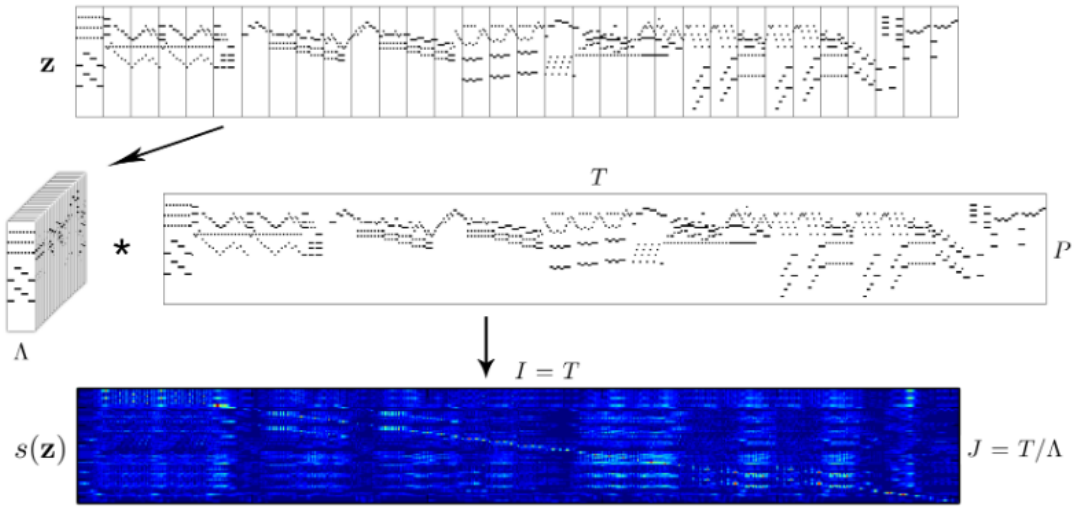


Fig. 9: Depiction of calculating the self-similarity matrix $s(\mathbf{z})$ (Lattner, 2019), $\mathbf{z}$ is the piano roll representation, and Λ the filter for convolution.

2. the tonality constraint: the distribution of pitch classes in every segment ("window") of the piece is filtered by two "key profiles" (arrays of numbers expressing the strength of each pitch class, one for the major keys, one for the minor)
3. the meter constraint: the number of notes in every bar is calculated, normalized and then constrained to the same relative value of note onsets of a template piece.

3 GENERATIVE EVOLUTIONARY ALGORITHMS

Evolutionary computing is based on Darwinian search algorithms inspired by natural evolution. These algorithms use processes analogous to natural selection, mutation, and reproduction to perform searches (Husbands, 2007). They operate on the concept of a population, which is a set of candidate solutions that evolves iteratively from one generation to the next. In evolutionary algorithms, each new generation is created by selecting the best candidate solutions (or "individuals") according to a fitness function. Additionally, new individuals in each generation are produced through genetic mechanisms such as mutation, crossover, and selection. These mechanisms are implemented in various ways depending on the specific type of algorithm.

Alan Turing suggested using the principle of evolution to develop adaptive machines as early as the 1950s. However, it wasn't until the late 1980s that this concept, particularly in the form of genetic algorithms, saw widespread application across numerous fields, mainly as an optimization algorithm (Husbands, 2007)..

3.1 Evolutionary algorithms overview

Evolutionary algorithms are composed of genetic programming, genetic algorithms, evolutionary programming, and other algorithms, that use the concept of selection of individuals according to fitness or objective functions. The general algorithm of an evolutionary procedure (Fig. 10) is presented below (Bagavathi, 2019):

1. Select initial population $x_i = \{x_{i0}, x_{i1}, x_{i2}, ..., x_{iN}\}$
2. Determine the value of objective function $f(x_0)$ for each individual
3. Perform selection of the best individuals (which have the highest values of the objective function)
4. Perform crossover of the selected individuals according to some probability value
5. Perform mutation of the newly generated individuals with some probability
6. Repeat this process until the requested termination conditions are met

Typically, these algorithms use two representations of the individuals: the genotype and the phenotype. The genotype is utilized during recombination and mutation processes, while the phenotype represents the solution form of the individual. The phenotype is evaluated to determine how close the individual is to the desired solution. During selection, the phenotype is "translated" back into its genetic form, the

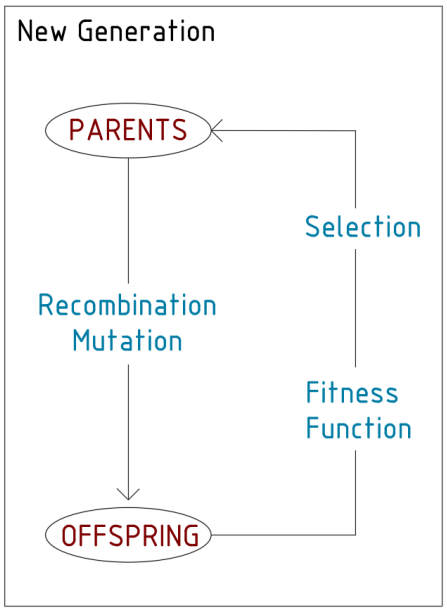


Fig. 10: Scheme of one generation of an Evolutionary algorithm

genotype, which is more practical for genetic procedures such as recombination and mutation.

3.1.1 Genetic representations

The genetic representation of candidate solutions to a problem can take various forms. For instance, if the solutions are numerical, their bit representation can be used. More commonly, genotypes are strings of numbers or symbols that unequivocally represent a possible solution (Husbands, 2007). For representing music and creating music algorithms, two main approaches can be chosen. The first approach uses strings of real numbers to represent sound characteristics for later sound synthesis. This method is optimal for applications where classical music structures and elements (such as notes, rhythms, and phrases) are not used, leading to the creation of new sounds or patterns.

Conversely, when the goal is to generate standard music melodies, fragments, or pieces, an approach mirroring the structure of MIDI files can be used. In this case, each note is represented by a string of numbers, which includes information about its pitch, duration, and dynamics, among other attributes.

In some cases, the genetic representation of the solution is equivalent to or very similar to the candidate solution itself. In such instances, the fitness evaluation of these solutions can be performed without translating the genotype into its solution form (the "phenotype").

The representations of individuals form the genotype space in which the best solution is sought. One of the strengths of evolutionary algorithms is the wide range of possibilities for representation. However, it is important to note that not all genotype spaces lead to feasible solutions.

3.1.2 Fitness (object) functions

After generating a set of candidate solutions, the next step is to select the best ones to become parents for the new generation. The selection process identifies the top individuals from the set based on a "fitness value," which indicates how close each individual is to the desired solution. This fitness value is typically derived from a fitness or objective function defined within the algorithm according to the optimization goal.

The simplest way to evaluate the solution's effectiveness is through a mathematical function, with variables directly encoded in the genotypes (Husbands, 2007). In other cases, it is necessary to build a model of the phenotype and then assess its value.

In the field of music generation, fitness evaluation is one of the most crucial steps. Sometimes, this evaluation is performed subjectively by musicians, but more often, an automated evaluation is attempted. Individuals can be assessed by comparing them to a target solution or analyzing various aesthetic and musical parameters. Using human judgment as the fitness measure, known as aesthetic selection or interactive evolution (Husbands, 2007), is problematic because it is time-consuming, subjective, and hence possible only for a limited number of generations. The selected individuals act as parents to the next generation.

After the selection process, the creation of a new generation individuals takes place. Depending on the algorithm, some or all individuals in the current population may be replaced by offspring. In spatially distributed populations, only individuals within a certain neighborhood might interact during the genetic operations, affecting the choise of individuals to replace.

3.1.3 Selection methods

After a fitness evaluation is made, there are different ways to select the individuals for the new generation. Usually, a probability element is added in this process, so the fittest individuals ("parents") are more likely to pass their information to the new ones (their "offspring").

An method that is widely implemented is the roulette selection, where each member of the population is assigned a probability of selection based on its relative fitness value (its fitness value divided by the total population fitness) (Husbands, 2007): the parents are then selected according to these probability values. This

mechanism sometimes converges prematurely to a solution that is just relatively optimal.

Another selection method is the rank-based selection: the population is ordered according to the fitness values and selection is then performed following a pre-determined probability distribution (Husbands, 2007).

3.1.4 Genetic operators

Variation, continual improvement and innovation when creating new individuals are secured by genetic operators, which are specific for every algorithm. The two operators that are used almost in every procedure are crossover and mutation.

Crossover involves choosing one or more points and creating a new individual (offspring) as a combination of different parts of the preexisting individuals (parents). These parts are made by dividing parents using crossover points. The number of children created by two parents can change, but the simple crossover gives two offspring individuals from two pre-existing ones.

Mutation on the other hand takes a randomly chosen element (gene) of an individual and changes it according to a probability value. The value of the mutating element can vary according to the mutation typology implemented. For example, it can take any value from all the possible values of genotypes or a value close to the original one.

Other operators are for example inversion, which is an operator that reverses a section of a genotype, translocation, which moves parts of a genotype to another place or deletion, when the length of a genotype needs to be shortened.

3.2 Genetic Algorithm

Genetic algorithm (GA) is a population based search algorithm, where a new population is iteratively created by selecting the fittest elements of the previous one. To do so, the algorithm iteratively applies genetic operators on individuals present in the population. These operators are: selection, crossover, mutation, and fitness function computation. Each element of the population is called chromosome and has often a binary form (Katoch, 2021). The binary encoding provides faster implementation of the operators, but it is appropriate only when the conversion from candidate solution to binary strings is not computationally too complex.

When a population is initialized each element of the population is evaluated according to the fitness function. The fittest individuals are then selected and variated by the genetic operators of crossover and mutation. The number of offspring individuals obtained by this variation process can vary. Not all individuals are vari-

ated at every iteration: the crossover and mutation happen only according to a probability value that is defined during the initialization of this algorithm.

The initialization of the population can occur in different ways (see Fig.11), but the most usual one is a random initialization (Kochenderfer, 2019).

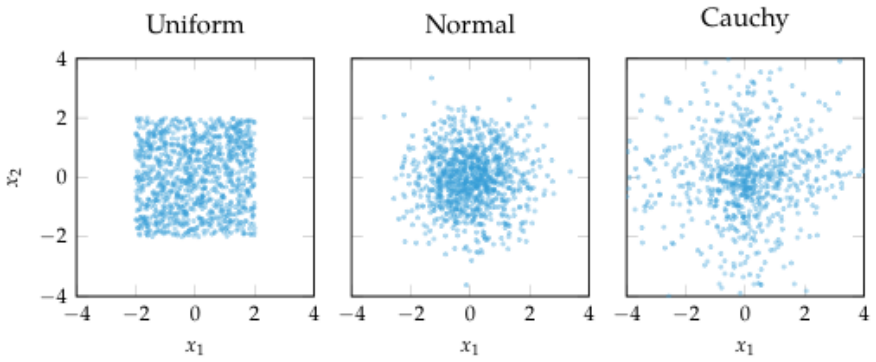


Fig. 11: Initial population using a uniform hyperrectangle, a zero-mean normal distribution and Cauchy distribucion ($C = [0, 0], \sigma = 1$)

3.3 Probabilistic grammatical evolution

Grammatical evolution (GE) is an evolutionary algorithm that employs grammars as the central element of its implementation. Grammars define how elements (such as programs) can be constructed from constituent parts by specifying how variables and operators can be legally combined to create elements with desired characteristics, like executable code (Ryan, 2018). According to Chomsky's theory, there are different types of grammars, each capable of producing specific types of languages. Most evolutionary computing systems, including GE, use Context-Free Grammars (CFGs), which correspond to Chomsky's Type-2 grammars.

Secondly, grammatical evolution represents genotypes in the form of linear strings (binary or integers). These are "mapped" to phenotypes, which are actual possible solution to the problem, according to the defined grammar's tules. In the grammar the relationships between variables and operators are specified. Therefore, a grammar-based approach enables to constrain the solution space, making easier the control over the creation process (Fredericks, 2023).

These algorithms have a modular nature, thanks to which they can be adapted to any kind of application. Each structural unit of GE can be set as needed (Fig.12): the fitness function, grammar, search engine and the mapping can all have any form (Ryan, 2018).

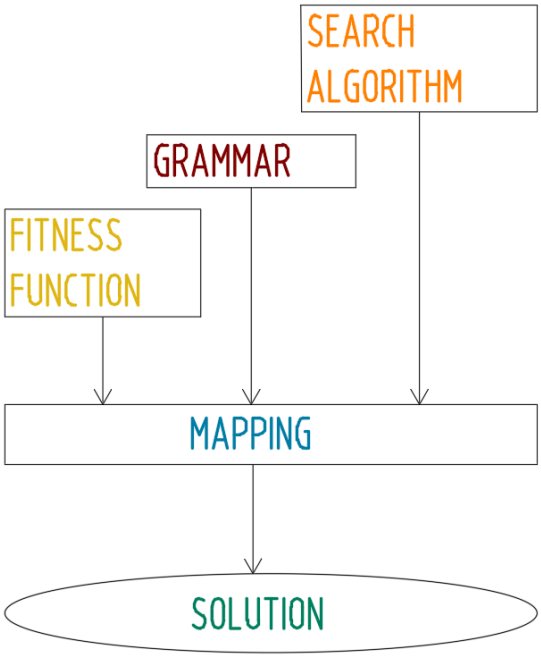


Fig. 12: Modular GE scheme from (Ryan, 2018)

3.3.1 Grammar

Grammars are a set of rules and symbols that determine how the resulting elements will look like. To define a grammar, there must be an "alphabet" V, with all the symbols necessary to build a grammar and sequences according to its rules. The set V is called free monoid generated by the set V and consists of all finite sequences of symbols (words, strings, denoted by v_i'') in V , that is (Pettorossi, 2022):

$$V = \{v_1, v_2, ..., v_n \| n \geq 0 \text{ and for } i = 0, 1, ..., n, v_i \in V\} \tag{1}$$

Moreover, the set V^+ consists of all non-empty sequences from the alphabet V, that is $V^+ = V - \{\varepsilon\}$. A grammar is defined as 4-tuple $G = \{T, N, P, S\}$, where (Pettorossi, 2022):

- T is the set of terminal symbols, which are part of the solution after parsing
- N is the set of non-terminal symbols, such that $T \cap N = \varnothing$
- A set of productions or rules P, each pair $[\alpha, \beta]$ being denoted by $\alpha \rightarrow \beta$, where $\alpha \in V^+$ and $\beta \in V^*$, with $V = T \cup N$
- A start symbol S or axiom

Each grammar defines a language L: a set of all sequences of terminal symbols, derived from the starting symbol S.

$$L(G) = w | S \rightarrow_G^* w, \text{ where } w \in T^* \tag{2}$$

where $\rightarrow_G^*$ represents the reflexive, transitive closure of the relation $\rightarrow_G$, defined as (Pettorossi, 2022),:

$$\text{for every sequence } \alpha \in V^+ \wedge \beta,\ \gamma\ \delta\ \in V^* \quad (3)$$

$$\gamma\alpha\delta\ \rightarrow\ \gamma\beta\delta \quad \text{if there exists a production } \alpha \rightarrow \beta \text{ in P} \quad (4)$$

A language is defined as Context-Free, if it is of type 2 according to the Chomsky Hierarchy, that means that for every production rule of that language:

$$\alpha \rightarrow \beta, \quad \text{where } \alpha \in N \text{ and } \beta \in \mathrm{V}^+ \quad (5)$$

For example, a Context-Free grammar $G = \{NT, T, P, S\}$ can be defined as follows:

$$\begin{aligned} NT &= E, O, E, N \\ T &= 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, \div \\ S &= N \end{aligned} \quad (6)$$

The production rules P of this grammar are:

$$\begin{aligned} E &\rightarrow EOE|N \\ O &\rightarrow +|-|*|\div \\ N &\rightarrow 0|1|2|3|4|5|6|7|8|9 \end{aligned} \quad (7)$$

Languages created by Context-Free grammar are usually presented in the Backus-Naur form (BNF), which is a specific syntax for defining production rules. For example the Type 2 grammar from Eq.7 will have the following BNF form:

$$\begin{aligned} &< Start > &&::= < Expr > \\ &< Expr > &&::= < Expr >< Op >< Expr > | < Num > \\ &< Op > &&::= +|-|*|\div \\ &< Num > &&::= 0|1|2|3|4|5|6|7|8|9 \end{aligned} \quad (8)$$

The Backus-Naur form is used in programming languages when implementing grammars.

In Probabilistic Context-Free Grammars, to these elements in the tuple is added a set of probabilities "Probs" associated with each production rule: $G = \{T, N, P, S, Probs\}$. These probabilities are changed after every generation, according to the rule's usage in the previous ones. The rules used to create the sequences that have better fitness scores, will have in the next generation a higher probability value (Mégane, 2022) depending on the chosen value of the learning factor.

3.3.2 Genotype to phenotype mapping

The mapping process is made of derivation steps and can be visualized using syntax trees.

1. A genome in the form of string of characters (codons) in the input to the mapping process
2. The codon will be replaced by that rule, which is in the place number resulting from the operation: $\mathrm{mod_n}(\mathrm{codon})$, where n is the number of available rules for that symbol. For example, if the codon is 45 and an operator from the grammar defined in Eq. 8 has to be chosen, $\mathrm{mod}_4(45) = 1$, which means we'll chose the second rule, in this case the operator <->.
3. This process continues, consuming a codon for each choice.
4. If there are still rules to be mapped and no codons available, new ones will be added, or that individual is abandoned (this is called the wrapping process).

The mapping process for Context-Free grammars can be represented as a derivation tree, with every derivation of each symbol (word) from the Start symbol S (axiom).

As an example, consider the following derivation from the grammar presented in Eq. 8 (Mégane, 2022):

Codons	Derivation
$-$	$< Start >$
$12\mathrm{mod}_1 = 0$	$< Expr >$
$46\mathrm{mod}_2 = 0$	$< Expr >< Op >< Expr >$
$35\mathrm{mod}_2 = 1$	$< Num >< Op >< Expr >$
$22\mathrm{mod}_{10} = 2$	$2\ < Op >< Expr >< Op >< Expr >$
$15\mathrm{mod}_4 = 1$	$2 -\ < Var >< Op >< Expr >$
$88\mathrm{mod}_{10} = 8$	$2 -\ < 8 >< Op >< Expr >$
$52\mathrm{mod}_4 = 0$	$2\ -\ 8 +\ < Expr >$
$27\mathrm{mod}_2 = 1$	$2\ -\ 8 +\ < Num >$
$97\mathrm{mod}_{10} = 7$	$2\ -\ 8\ +\ 7$

(9)

The parsing tree diagram for this derivation process is presented in Fig.13:

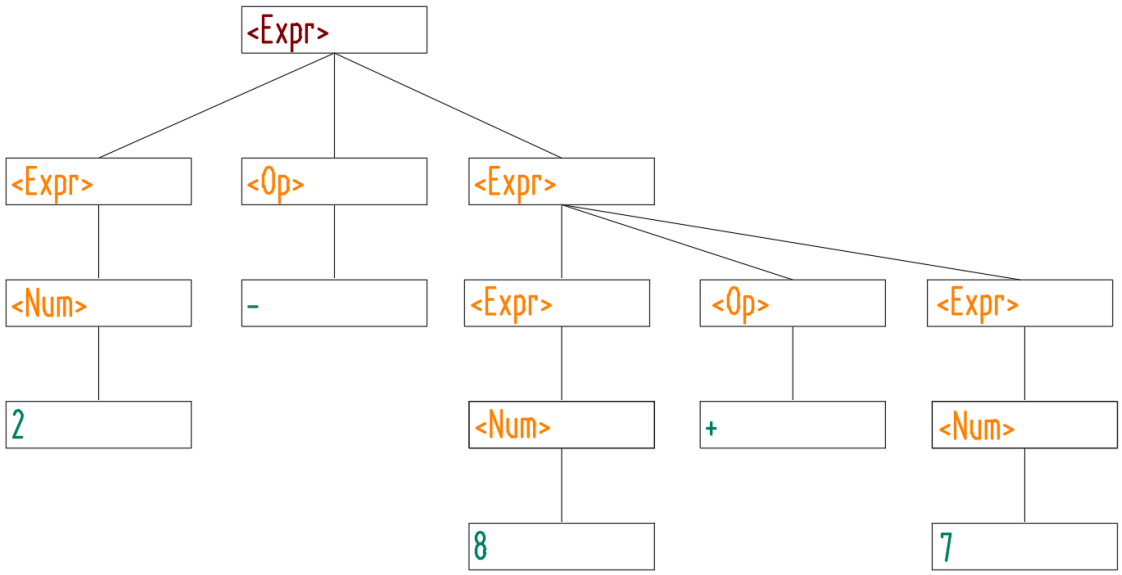


Fig. 13: Derivation tree for the mapping process

When to this grammar probabilities are added, the mapping won't be based on the result of the mod operation, but on the probability range of each rule. Therefore, initially to every production rule a probability range is assigned. For example, if there are two rules and we want to use the first one with 80% of probability, the probability ranges will be $(0; 0, 8]$ for the first one, and $(0, 8; 1]$ for the second one. The Probabilistic Context-Free equivalent to the grammar in Eq.7 will have the following form:

$$
\begin{array}{ll}
\text{Rules} & \text{Probability ranges} \\
E \rightarrow EOE|N & (0; 0, 7] \,|\, (0, 7; 1] \\
O \rightarrow \ + \,|\, - \,|\, * \,|\div & (0; 0, 25] \,|\, (0.25; 0, 5] \,|\, (0, 5; 0, 75] \,|\, (0, 75; 1] \\
N \rightarrow 0 \,|\, 1 \,|\, 2 \,|\, 3 \,| & (0; 0, 05] \,|\, (0.05; 0, 3] \,|\, (0, 3; 0, 45] \,|\, (0, 45; 0, 47] \,| \\
\quad 4 \,|\, 5 \,|\, 6 \,|\, 7 \,| & (0, 47; 0, 5] \,|\, (0, 5; 0, 6] \,|\, (0, 6; 0, 75] \,|\, (0, 75; 0, 85] \,| \\
\quad 8 \,|\, 9 & (0, 85; 0, 9] \,|\, (0, 9; 1]
\end{array}
\tag{10}
$$

3.3.3 Search engine

The choice of the search engine can determine the quality of the algorithm implementation. There are multiple possibilities that have been used, from traditional evolutionary algorithms like Particle Swarn Optimisation, Simulated Annealing, Differential Evolution, to random search or geometric semantic searches (Ryan, 2018). The search engine comprehends an initialisation of the data, an optimisation algorithm and genome encoding. In the case of evolutionary algorithms the behaviour

of search operators (e.g. crossover, mutation) is crucial, therefore needs to be set to best meet the needs of the specific application.

3.4 Application in music

In music, evolutionary algorithms have been experimented with in two main areas: music composition and sound design. In music composition, these algorithms can continue pre-existing melodies, compose pieces in specific styles (e.g., Bach's cantatas), or generate entirely new material. Music theory itself is inherently algorithmic, relying on harmonic rules, counterpoint, and phraseology. However, these rules cannot be entirely translated into machine learning algorithms because music also relies on the creative ideas that are fundamental to each piece.

The strength of evolutionary algorithms lies in their controllable and guided generative processes, making them suitable for encoding musical rules akin to a composer's work. However, there is a risk that these algorithms might produce overly rule-based outputs, resembling composition exercises rather than genuine musical compositions. While this approach can be useful as a tool to aid composers by handling some of the craftsmanship that can slow down the creative process, it can stifle creativity if the aim is to produce new music. To create truly original music, it is essential to preserve elements of "the unexpected," a "sense of direction," and "fantasy." The explorative and stochastic nature of evolutionary algorithms can facilitate this creative process by not strictly defining the search space (Husbands, 2007).

Musicological studies have analyzed composition rules, forms, and schemes from various historical periods. Until the early 20th century, composers used these predefined forms as a foundation for creative experimentation. This historical context makes it challenging to use these musical forms as ultimate models for evolutionary algorithms.

In sound design, evolutionary algorithms are used for sound synthesis or effects modification, aiming to create new sounds or optimize existing ones. However, these applications often demand high computational resources and may yield suboptimal results, leading to the preference for neural networks. In this domain, sound spaces can be explored either within constraints or freely, depending on the desired outcome.

A common issue reported by practitioners of computer-based music approaches is a lack of overall musical energy or flow, resulting in a lack of global coherence (Husbands, 2007). This problem arises because these algorithms are not inherently time-based. One potential solution is to introduce rules that create long-term rela-

tionships and spatiotemporal structures, such as using a chemical oscillator rule in a cellular automata model (Miranda, 2000).

4 NEURAL NETWORKS

Neural networks are employed to simulate various functions according to defined objectives. Presently, much of the research on neural networks used in music composition focuses on computationally and memory-intensive tasks such as data preprocessingâĂŤsuch as labeling and data cleaning (Jin, 2020).

As observed in Chapter 2.2, over the last decade, many neural networks designed for music composition have integrated architectures like generative adversarial networks (GANs), long short-term memory networks (LSTMs), convolutional networks (CNNs), actor-critic (AC) networks, among others. In the subsequent sections, we delve into the fundamental structures of these networks, elucidating how music composition benefits from their specific characteristics.

4.1 Generative adversarial networks (GAN)

Generative adversarial networks (GANs) are capable of learning high-dimensional distributions, which are challenging to model explicitly due to their complexity. Instead, GANs learn these distributions implicitly from the provided datasets, such as images, audio, and more. A basic GAN consists of two neural networks that aim to optimize opposing loss functions. Therefore, the main components of these networks include the network architecture, the loss function, and the optimization algorithm (Saxena, 2021)..

The two networks employed are the generator and the discriminator: the former crafts samples, while the latter distinguishes between generated samples and real data, as depicted in Fig.14. Additionally, two distinct loss functions are utilized. The discriminator's loss function minimizes the negative log-likelihood for binary classification, whereas the generator's maximizes the likelihood of the generated samples being deemed authentic (Saxena, 2021). Optimization entails tackling a minmax problem, often addressed using gradient-based algorithms, particularly Simultaneous Gradient Descent. These networks collapses if the min-max solution works differently than the max-min: this way the generator creates samples that are constantly rejected by the discriminator.

In recent times many alternatives to the basic GAN model have been proposed: new architectures to make the training easier, different loss functions to reach a better parameters' stability and convergence and sometimes also for the optimization algorithm a different gradient descent method is used (Saxena, 2021).

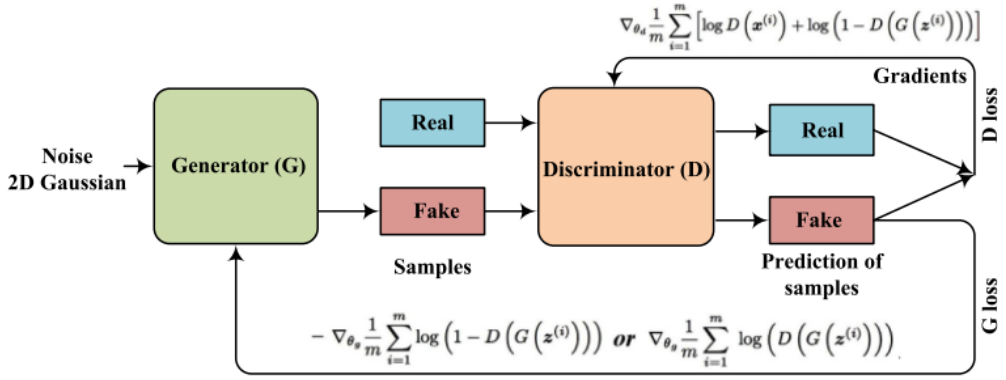


Fig. 14: Basic GAN architecture (Saxena, 2021)

4.2 Long short-term memory networks (LSTM)

Long-short term memory networks can generate music sequences based on probability values: they are used in combination with CNN networks, that work as discriminator (Jin, 2020). These networks are a modification of RNN networks that can keep constant memory over a longer period of time thanks to the gating mechanism (Fig.15). An LSTM cell separates the previous output (h_t) from its memory state (the cell state, c_t) (JÄŹdrzejewska, 2018). The cell state can only be modified at the forget gate, where previous values are selectively discarded, allowing new data to be added. This addition of new data is managed by the input gate, which controls which new candidates will be incorporated to prevent frequent modifications of the cell state. In the final gate, the output gate, the cell state value is normalized using a hyperbolic function, and a mask determines the final output of the LSTM block for the given time step. This output also serves as the hidden state for the next time step (JÄŹdrzejewska, 2018).

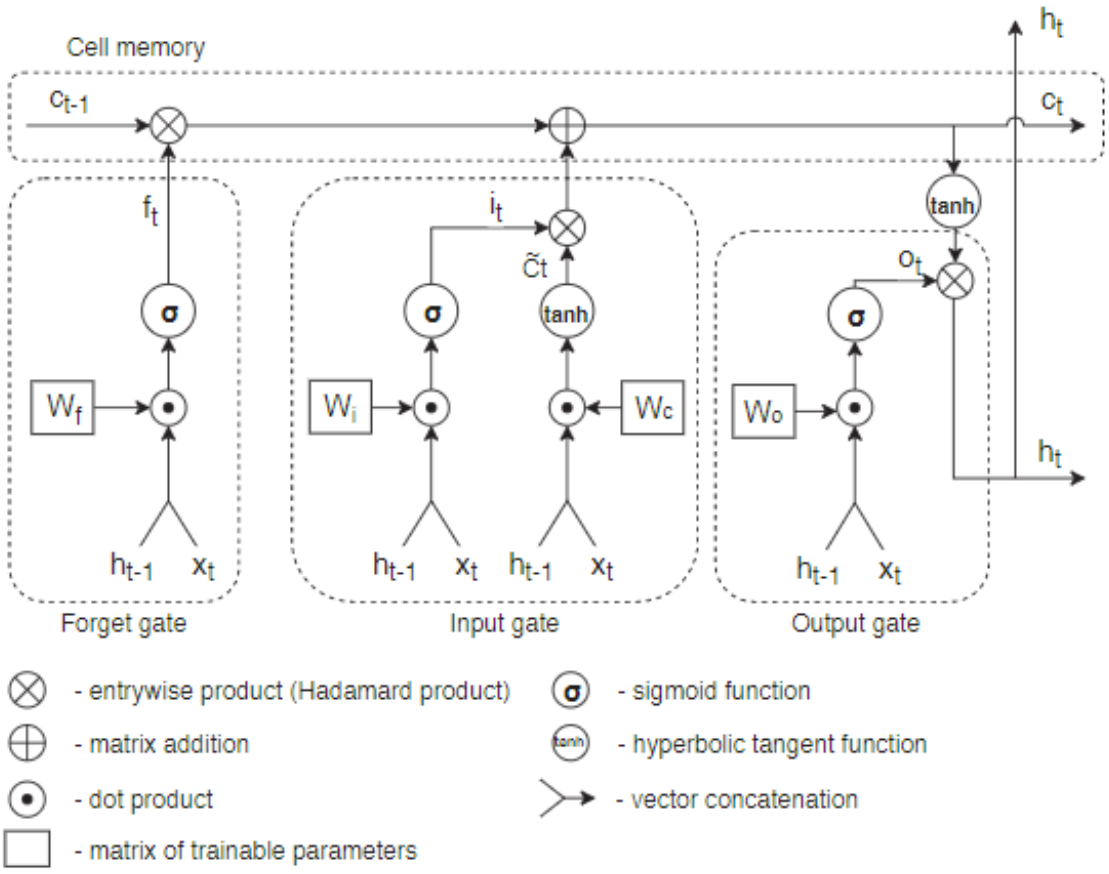


Fig. 15: Scheme of the gating mechanism of a LSTM network (Jędrzejewska, 2018)

5 THE MUSIC COMPOSITION PROCESS

5.1 Artistic composition

The Czech composer Leoš Janáček described the composition process as perceived by the music composer as involving assimilation, apperception, association, and reproduction. Assimilation involves perceiving a sound in nature, which is a blend of many tones creating its timbre. Once the composer assimilates this sound, they internalize it by noting its length (rhythmic value) and experiencing its beginning and end. Next is the process of association, where this note is linked with the staff and a key signature. Reproduction involves playing the note in its key, enabling the composer to associate it with the staff, potential movements, and a key signature.

Composing music, therefore, is a process of internalization and association. Internalization begins with whatever captures the composer's attention or imagination. Following this is the process of reproduction, which can be based on previous creations (musical interpretation) or the creative reproduction of the composer's associative process. Composition is thus based on compositional notes that express emotions, an "affective stream," and simultaneously react to these stimuli. According to Janáček, composing should merely visualize moods and movements, with chords seen as symbols: "compositional work should achieve the speed of musical imagination," allowing moods alone to drive the composing process (Janáček, 2020).

When discussing music, we often use terms like "musical piece," "track," "opus," etc. Listening to or playing music is always contextualized by specifying which "part" of the music is being referred to. Music is divided into pieces, which are themselves structured into smaller sections, continuing this pattern. This division stems from a binary interpretation of music (one part "A" being different from another part "B"), leading to binary or ternary musical forms (e.g., ABA) (Cook, 1998). These parts, in any genre or historical era, are labeled with letters, numbers, or names reflecting linguistic syntax such as motives, phrases, and periods. In graph theory terms, almost every musical piece can be represented by a tree structure.

Western music notation, on the other hand, reflects the Cartesian system: it is a bidimensional representation of pitches over time. Thus, music appears to have two dimensions: a spatial one (describing its structure) and a temporal one (describing its flow in time). Correspondingly, music appreciation can also be considered to have two dimensions: one based on the sensory, temporal aesthetic value of a musical piece, and the other reflecting the aesthetic of musical form from a Kantian perspective (Cook, 1998). The latter, called "structural listening," reflects cultural and traditional listening habits. Charles Rosen refers to this as "inaudible music": musical form that is often not translatable into sound but is perceived through cre-

ative, musical imagination (Rosen, 1995). Janáček describes this significant aspect of music as its "residual sense," and the cohabitation of these two dimensions as a creative force, or "musical feeling" (Janáček, 2020). This is akin to the way the value of spoken words lies in their meaning, not their sound, with understanding coming from the intellect. Historically, many experiments have been based on this concept, from the notation of Ricercare by Carl Philipp Emanuel Bach in 1747 to the Inner voice written by Schumann in his Humoreske. Ignoring this internal experience would remove the "excitement" derived from listening to music (Janáček, 2020).

Theories from musicologists like Schenker or Rosen serve as "listening manuals," providing an aesthetic vocabulary for structural listening, typically applied to specific music genres. These theories can be used to compare different musical pieces and genres.

5.2 Musical analysis

Analyzing musical works involves asking questions like “How is a piece built?” and “How does a piece work?” (Mastropasqua, 1998). According to the musicologist Mastropasqua, music comprises "perceived structures" and "designed structures." These designed structures are part of the composer’s strategy but aren’t immediately perceived by the listener and need to be illuminated through analysis. The relationship between what the listener perceives and what the composer designs is a central theme in musical analysis, distinguishing contemporary from modern musical styles. Starting with the compositions of Schoenberg and Webern, the focus shifted from the perceived musical structures to the composer’s designs. Understanding the divergence and relationship between these structures is crucial for interpreting 20th-century music. Before the 20th century, there was a direct correlation between the music perceived by the listener and the structure intended by the composer, with the perceived music being a consequence of the underlying compositional structure (Mastropasqua, 1998).

In tonal music, this fundamental structure was described by H. Schenker, whose theory reveals a unique harmonic and melodic structure spanning the entire piece: the tonic (I) moving to the dominant (V) and then returning to the tonic at the end. Every composer and musical style expands and varies this structure but never abandons it. For example, in Fig. 16, two chord analyses of the first part of Bach’s C major prelude are presented, and in Fig. 17, Schenker’s fundamental structure of the same piece is shown. At the beginning of the 20th century, music composition took a different direction: atonality. Atonal music replaced the tension structure of tonic and dominant harmony with a structure of tensions and resolutions generated by choices of sounds, rhythms, or other “performative” effects.

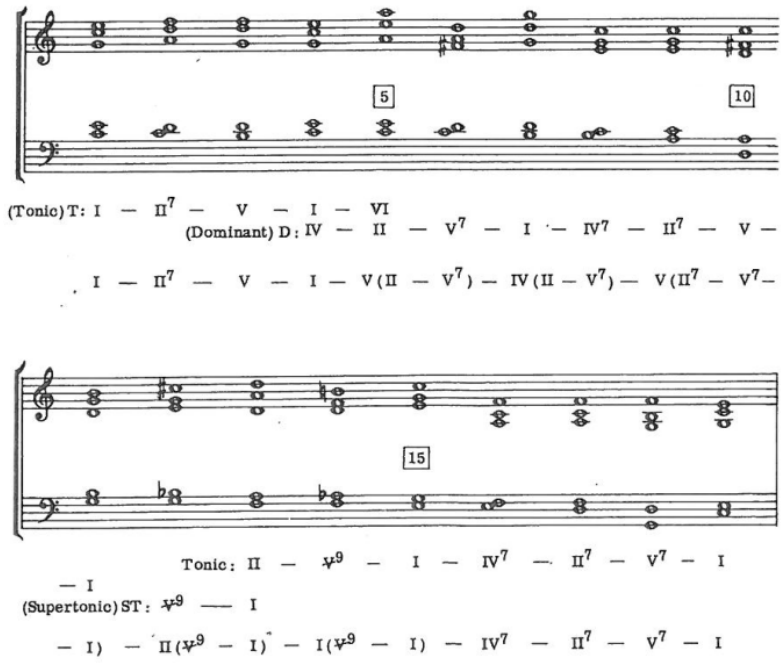


Fig. 16: Harmonical structure of Bach's prelude in C major (bars 1-19) (Cook, 1998)

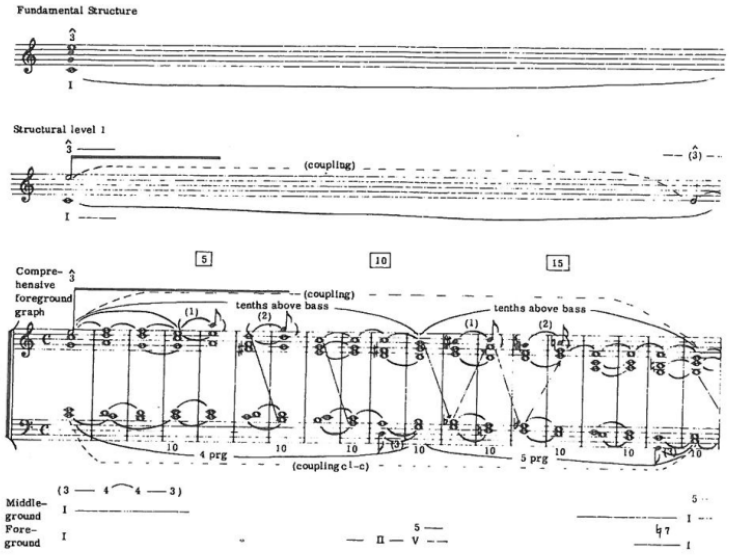


Fig. 17: Fundamental structure of Bach's prelude in C major (bars 1-19) (Schenker, 1969)

Structural analysis in tonal music is a deductive process, relying on a previously known form. In contrast, analyzing atonal music requires an inductive approach, where relationships must be discovered, often leading to subjective analysis based on potentially incorrect premises. Experiences analyzing atonal music, such as Webern's compositions, have shown that understanding a piece can come solely from listening, without the need for structuralism. However, searching for the structure of a musical piece offers an intriguing study of the relationship between the sensory and logical dimensions, which has fascinated many modern composers (Mastropasqua, 1998).

H. Schenker developed an analytical system to analyze music and define its "Ursatz," or fundamental structure. Schenker viewed a musical piece as a "large-scale embellishment" of a simple harmonic structure. As can be seen in Fig.18, his analysis comprises three main layers: the upper line (Urlinie) or foreground (Vordergrund), which resembles the score of the piece; the structural graph or middle ground, showing the movement of harmonies and melodies and describing how fundamental elements connect; and the deep background, or "Ursatz," reflecting the core harmonic and melodic structure of the piece (Cook, 1998).

The aim of Schenkerian analysis is to reveal how people listen to music, uncovering the structure in which a composition's unique qualities are hidden: the relationships between melodies, motives, ornaments, and the Ursatz in the background. This analysis is particularly effective for German and Austrian music of the 18th and 19th centuries (e.g., composers like Bach, Beethoven, or Schubert), but it doesn't fit as well for Italian, French, or Russian music of the same period, reflecting Schenker's aesthetic sensibility. Schenker searched for the main direction in musical pieces, the progressive harmonic and melodic movement towards a climax and resolution, which is not present in every musical genre. For example, Debussy's music creates static "sound environments," refusing to give tonal direction to many compositions and not seeking organic coherence through harmonic structure. Thus, Schenkerian analysis doesn't describe Debussy's compositions well (Cook, 1998).

The effectiveness of the Schenkerian method depends not on its principles but on the conventions the analyst declares before starting the analysis. The process of iteratively simplifying the structure to uncover its core relationships remains a valuable tool for musicologists. For instance, in Fig.**??**, an analysis of Webern's Bagatelle op. 6 uses Schenker's structure but starts from the concepts of tension and resolution rather than harmonic bonds between musical chords and phrases.

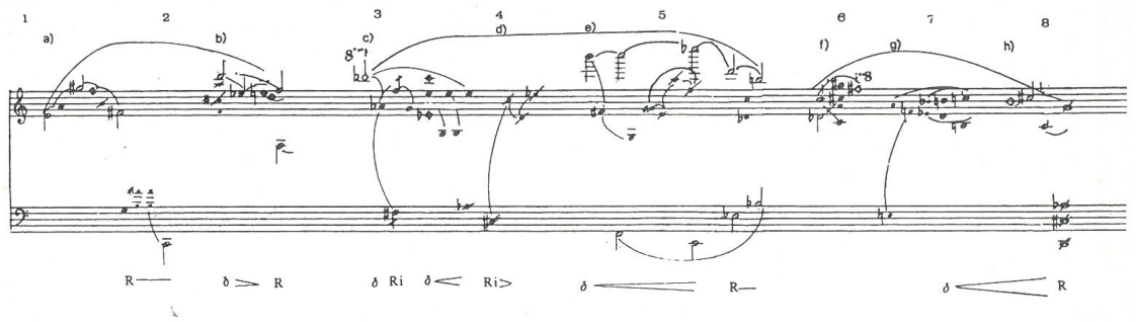


Fig. 18: The analysed structure of Webern's Bagatella n. 2 (Mastropasqua, 1998)

5.3 Melody synthesis: an automatic music composition's method

Janáček describes piano music composirion as a generation of justified rows of tones that become melodies thanks to their different strengths. He explains from a harmonic point of view how composers do not have to write complete chords, but only "melodies justified by dynamic relations" (Janáček, 2020). An example of an "arpeggiato" chord which makes a melody can be seen in Fig.19.

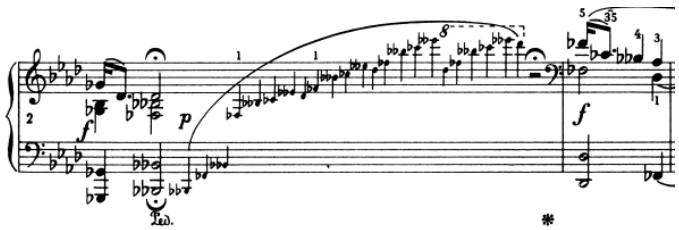


Fig. 19: Second measure of Chopin's Polonaise-Fantasie op. 61, an example of an arpeggiato chord (Chopin, 1846)

Row of tones make a voice when they possess some "musically expressive elements", which according to Janáček can be connected to the harmony, rhythm or dynamics used. Melodic voices can be less coherent, resolution, disturbance and appeasement often relate to other tones than the melodic ones, so they can be more free.

According to Riemann, melodic voices have ortography rules they need to fulfil to intensify impressions and feelings (Riemann, 2014). Janáček reacts to this concept of melody with a "new direction in music theory", which tries to associate the connecting forms in the melodies with the elements of musical syntax - the chord connections. According to Janáček harmonic resolutions are present in melodies too and become the "connecting forms" of the melody. For example an interval of minor

second, which played as a chord is dissonant, in a melody is a “resolution of the minor second to the unison”.

Chopin’s second’s piano sonata Finale is a good example of how a melodic voice creates connecting forms (harmonic connections) so strong, it stands on its own (Fig. 20).

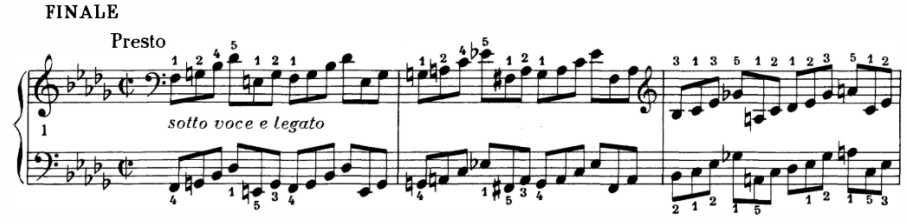


Fig. 20: Opening of the Finale of Chopin’s Piano sonata in b-flat minor op. 35 (Chopin, 1839)

Other fundamental principles of music harmony are outlined in JanÃąÄĎek’s new theory of music, which presents rules for connecting forms and relationships between intervals. For example, it states that “octaves are most effectively disturbed by major sevenths" and "major sevenths resolve most effectively into octaves.” According to Helmholtz’s scale of intervals, the consonance of intervals gradually diminishes (Fig. 21):

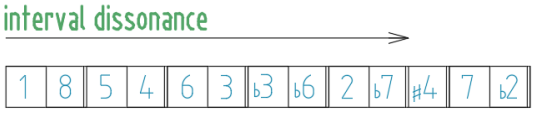


Fig. 21: The scale of interval consonance defined by Helmohltz (Janáček, 2020)

This theory of interval connections in melodies is incomplete without considering the rhythm: the character of connections between forms depends on the accentuation, given by the rhythmic shape of the melody. Rhythm gives movement to tones, Janáček says that every musical style has its own prominent figurations of “chaotic moments within connecting forms”, which he calls groupings of twine (in czech: “spletny”) (Janáček, 2020). One example of this twine can be seen in Chopin’s Polonaise (Fig.19), where the arpeggiato used to create a connection between distant tones and harmony.

6 APPLICATIONS

6.1 Melody structure: musical introduction

Melodies have two main components: pitches and time elements. The most common tuning system in western music is called the equal temperament: it divides the octave into 12 equal parts on a logarithmic scale. The frequence ratio between two adjacent parts is constant and equal to:

$$\frac{f_{x+1}}{f_x} = \sqrt[12]{2} \tag{11}$$

where f_x is the frequence of note and f_{x+1} is the frequence value of the next note, exactly a semitone (1/12 of an octave) higher. Thus, note frequencies form a geometric series and every note frequency can be calculated starting from the reference note. The reference is usually the A note at 440 Hz, but it can change according to the instrument in question. Hence the equation to calculate a note frequency is:

$$f_x = f_r \cdot (\sqrt[12]{2})^{x-r} \tag{12}$$

where f_r is the frequency of the reference note (in the case of a standard piano tuning it is 440 Hz) and r is the position of the reference note, in this case the A (440 Hz) is the 49th note of the piano, so $r = 49$ and x is the position of any other note of which we want to determine the frequency (for note positions on a piano keyboard refer to Fig. 22).

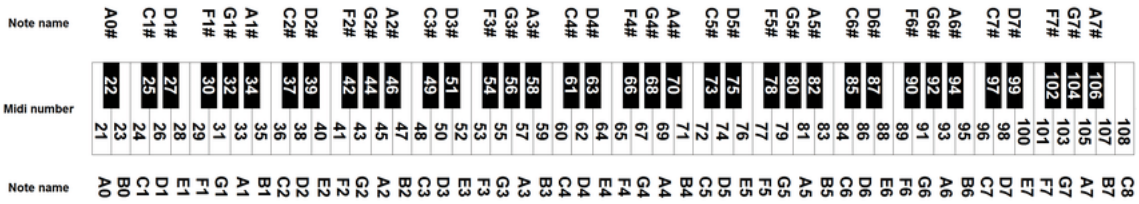


Fig. 22: Piano keys with their MIDI number

Frequencies vary for every instrument and tuning, they therefore don't univocally determine a musical note. When dealing with western musical compositions notes are usually defined by numbers. Nowadays the most common number notation for notes is the MIDI numbering, which is used in MIDI messages. Midi messages form ".mid" musical files, which are widely used for music reproduction. Fig. 23 represents the frequencies and MIDI numbers of the notes of a piano. In classical music the other instruments can play different ranges of these notes.

There are two time elements of a melody: the meter and the rhythm. Together, they build a succession of beats. The meter is equal for the whole melody

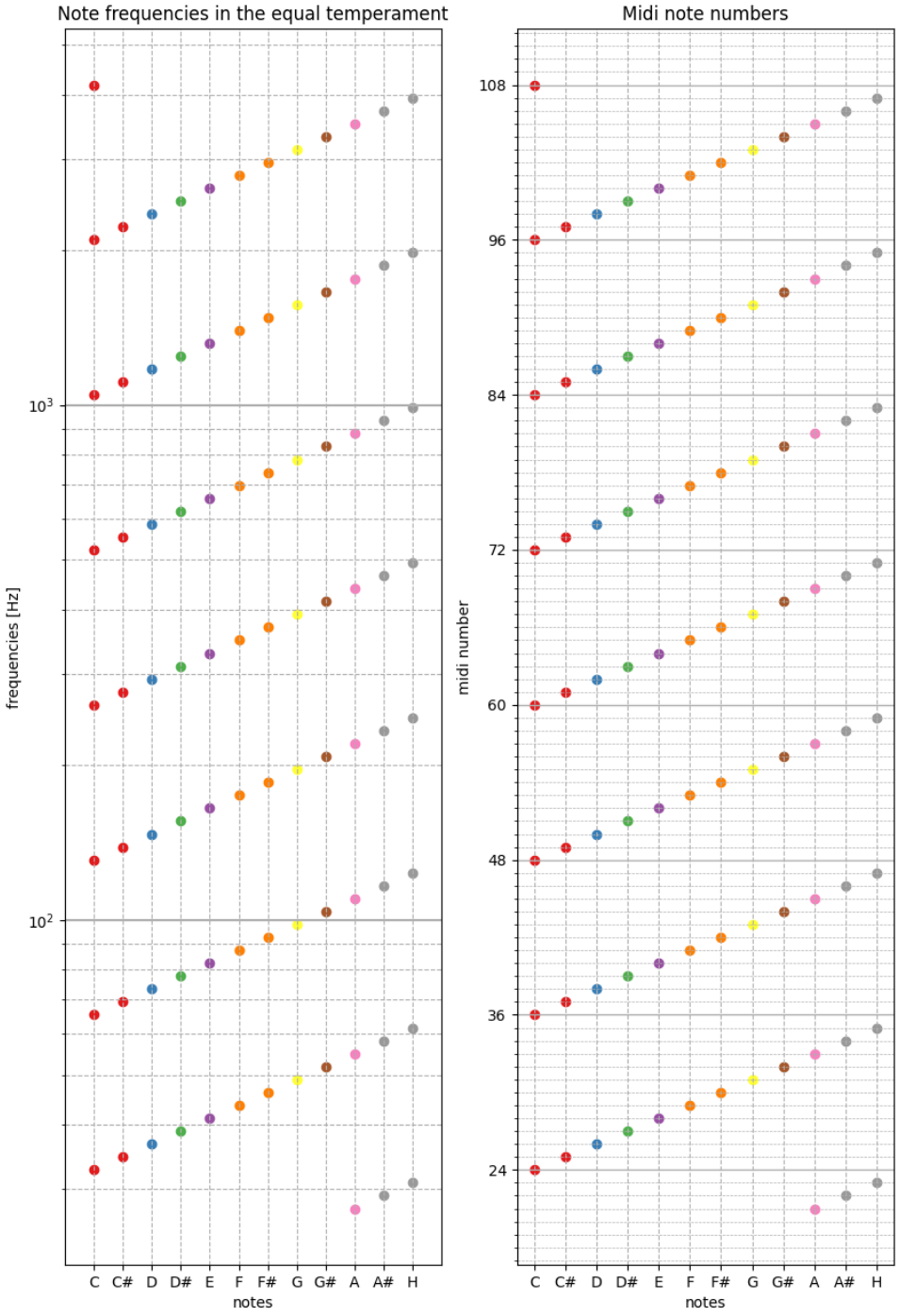


Fig. 23: Piano notes and their frequencies according to the equal temperament

(or for parts of it), it determines its accentuation. For example, a 3/4 meter can be used for the "waltz" accentuation, where there are 3 beats in every measure with the first one being the strongest one dynamically (the intensity of sound will be higher for the first beat).

Next to the "accentual shape" (Janáček, 2020) of a melody, there is also the "rhythmic" shape, which depends on the length of every note in the melody. The rhythm is therefore made of each note's length. In classical music these notes' lengths are categorized according to the number of beats (or the beat fractions) they last. The main notes' lengths and their names are shown in Fig. 24.

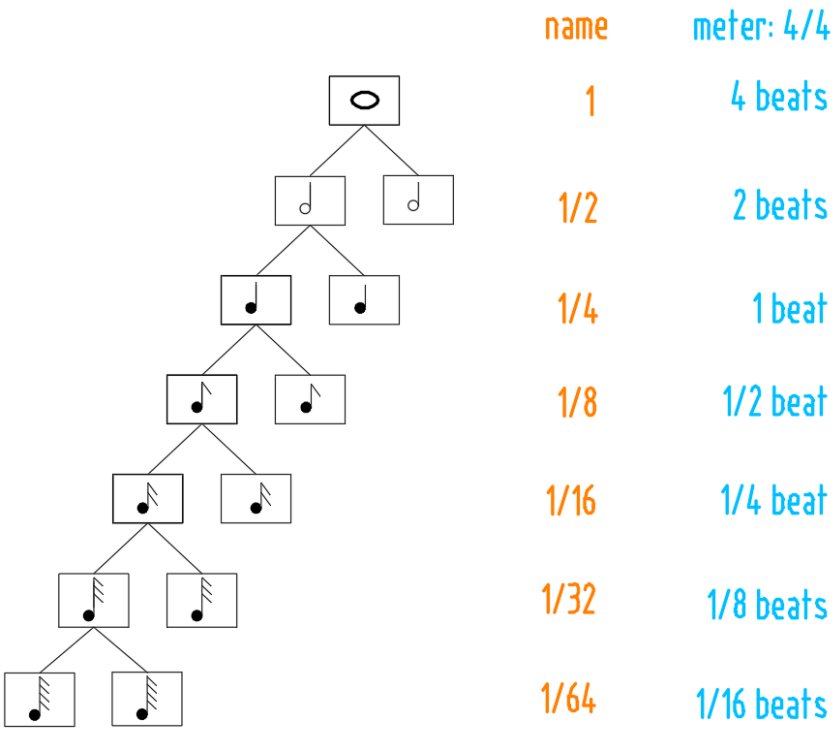


Fig. 24: Most common notes' lengths in classical music

In MIDI files, notes' lengths are written as the number of ticks that distance one event from the other (Fig. 25).

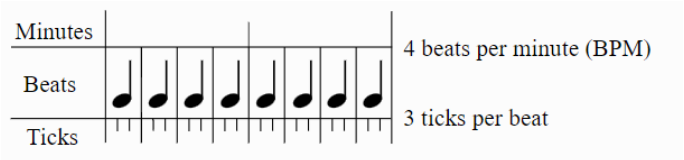


Fig. 25: Relationship between meter beats and MIDI file ticks (Bjørndalen, 2023)

The number of ticks in each bet is set at the beginning of the MIDI track. In Fig. 25 is presented another time measure, the BPM value, which determines the speed of the metric beats per minute.

6.2 Analysis Tool

A tool that extracts features from MIDI datasets was designed to visualize the main characteristics of a dataset and compare them to others. It was intended as a tool for composers and musical students, as well as for people without a musical education, that want to understand better the main differences between composer's styles and musical genres.

In this implementation, we use MIDI files, which include information about pitches, their durations, and the velocity with which they are struck, determining their intensity. Using the provided dataset, this tool analyzes the structure of each track and examines the following musical features.

6.2.1 Notes' density and variety

The first feature extracted from every track of the given dataset is the notes' distribution, as can be seen in Fig. 26, the density of the given note's occurrence is determined by the color at the note number position (x-coordinate). The y axis determines from which track the values are taken.

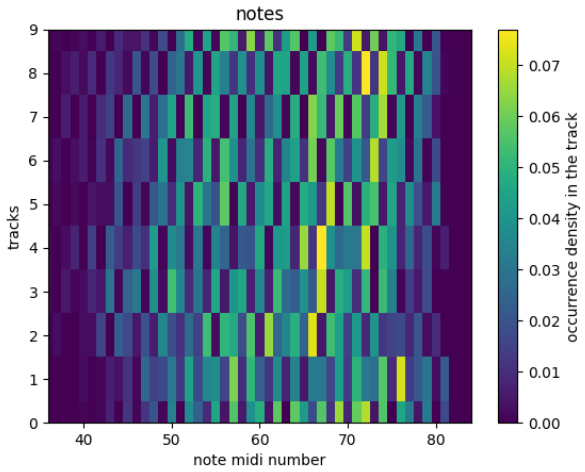


Fig. 26: Heatmap of the notes' density in each track

In Fig. 26 we can see that the notes' range and the notes' distribution, which can be seen in detail in the second graph resulting from the analysis, Fig. 27. This

histogram shows the mean distribution of each note, in other words the frequency with which they are repeated (proportional to the length of each track).

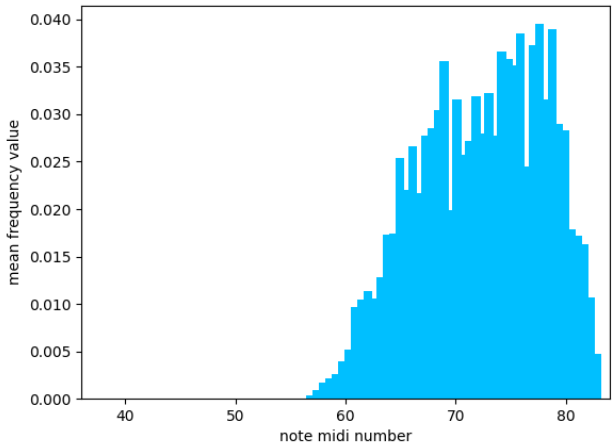


Fig. 27: Histogram of notes' mean distributions in the dataset

6.2.2 Interval density and variety

Besides the notes' range and occurrence, what defines a musical style is often the type of intervals that are mostly used. This is why the second feature extracted from the dataset is the distribution of intervals among the tracks.

Musical intervals determine the distances between notes, each interval name refers to a precise number of semitones, as can be seen in Tab. 1. The names of these intervals come from the tonality that they are in, i.e. if the given interval can be found in a major scale, we refer to it as a "major interval" and vice versa for minor scales. The intervals that have a minus or plus sign are augmented and diminished intervals. Note that the number of semitones in each interval corresponds to the difference between the interval notes' MIDI numbers. In Tab. 1 the intervals used in this work are showed.

Tab. 1: Table of used interval and the number of semitones they span

symbol	name	n. of semitones
I	Perfect unison	0
IImin	Minor second	1
IImaj	Major second	2
IIImin	Minor third	3
IIImaj	Major third	4
IV	Perfect fourth	5
IV+	Augmented fourth	6
V	Perfect fifth	7
VImin	Minor sixth	8
VImaj	Major sixth	9
VIImin	Minor seventh	10
VIImaj	Major seventh	11
VIII	Perfect octave	12

In Fig. 28 a density distribution of the intervals types is shown and in Fig. 29 the mean distribution values of each interval's occurrence are represented.

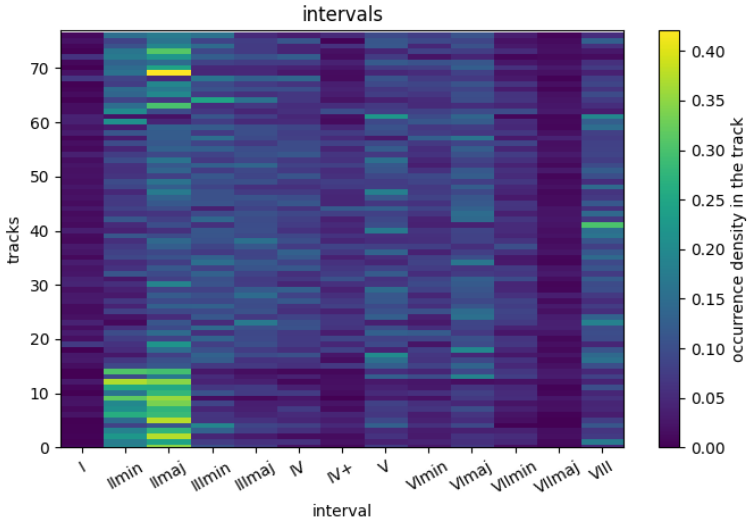


Fig. 28: Heatmap of the interval densities

An alternative representation of the most used intervals in the dataset, without considering the modality (if they are major or minor) can be seen in Fig. 30.

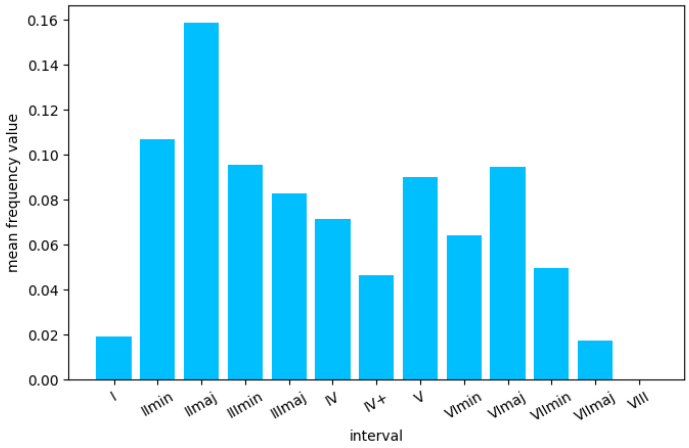


Fig. 29: Histogram of intervals' mean distributions in the dataset

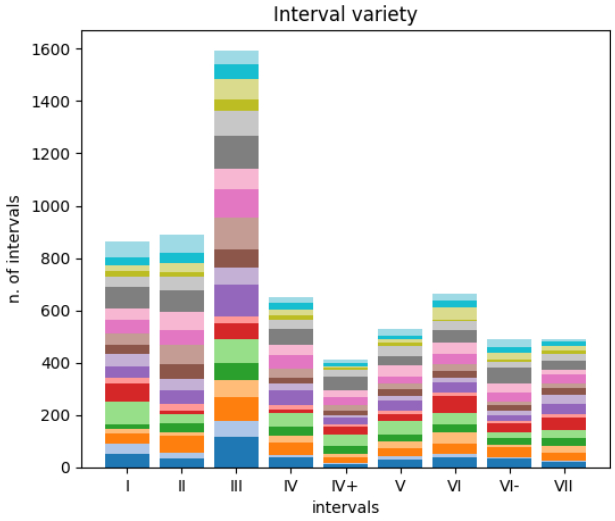


Fig. 30: Interval types and their usage in the dataset

Lastly, the number of melodic patterns is analyzed: the histogram in Fig. 31 shows the number of repeating patterns found in every track. As a melodic pattern a 4-notes melody segment is considered: if another same four notes segment is found in the track, the number of repeating patterns for the given track is increased by one.

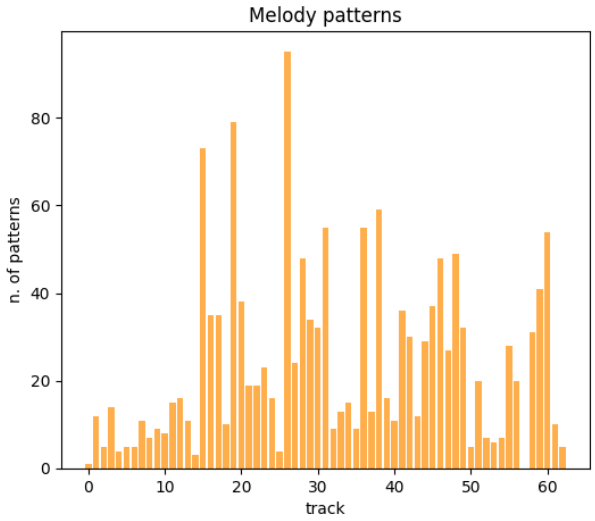


Fig. 31: A histogram showing the number of melody patterns

6.2.3 Notes' durations density and variety

As seen before, what makes a melody is also its rhythmic outline, which is the succession of note durations and metric accents. Metric accents depend on the meter of the piece and mostly don't change throughout the piece. Therefore, attention is paid to the notes durations. The durations' names used in Fig. 32 and Fig. 33 are described in Fig. 24. In the given dataset there are mostly notes that last a 0,0625 fraction of a beat, corresponding to a 1/16, a sixteenth note.

The rhythmic variety can be seen more clearly in the "Rhythmic variety" plot (Fig. 34), where the number of different notes' duration per track is plotted. The "Rhythmic patterns" plot (Fig. 35) of the same dataset confirms what can be seen in Fig. 34, as a high number of rhythmic patterns means a lesser variaty of notes' durations. The length of the parameters considers can be set according to the analyzed musical style and the length of each track.

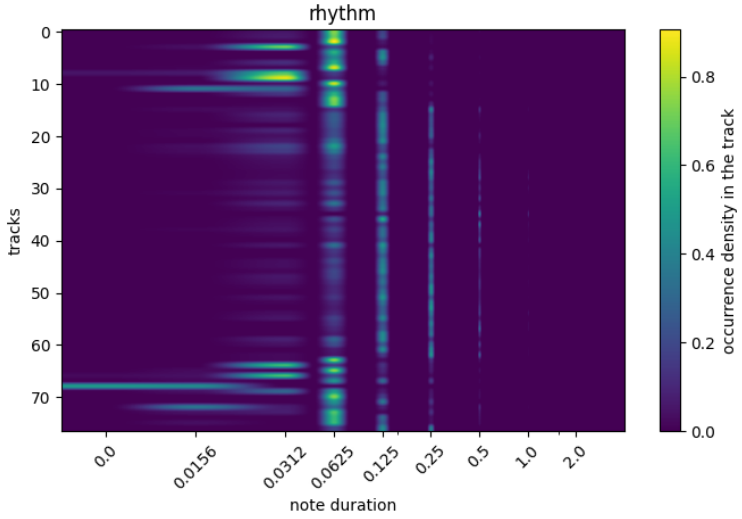


Fig. 32: Heatmap of the durations' density in each track

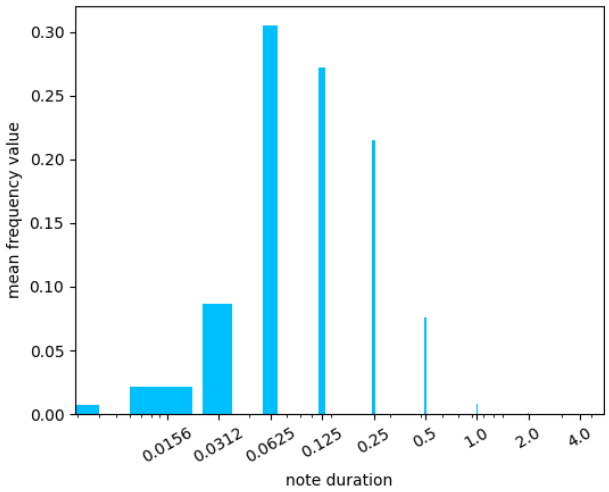


Fig. 33: Histogram of durations' mean distributions in the dataset

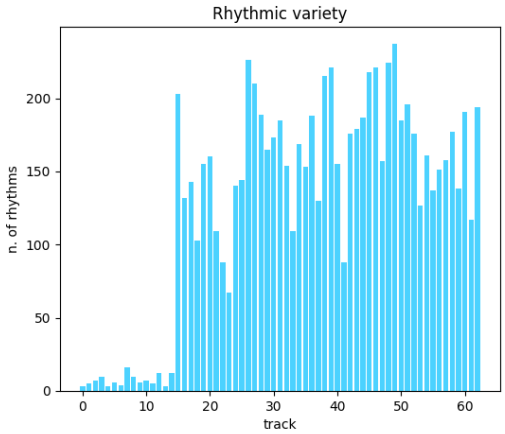


Fig. 34: Rhythmic variety plot

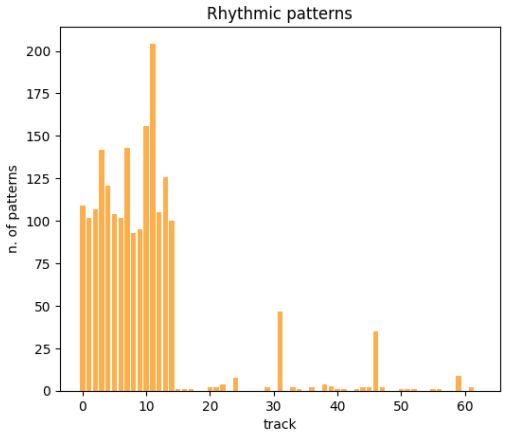


Fig. 35: Rhythmic patterns plot

6.2.4 Macrostructures

Tonalities are the core of the equal temperament system and were fundamental to every classical music piece from the Baroque era to the 20th century. Therefore, it was essential to develop a tool capable of extracting information about tonalities from the given datasets. Tonalities are closely linked to harmony rules and polyphony, which are beyond the scope of this work. As a result, an alternative approach to simulate tonality in simple monophonic melodies was necessary. It was decided to identify the tonality by considering the most recurrent pitch in a phrase (a melody segment of 4 measures), provided that the other most recurrent pitches were part of the same tonality's chord.

For example, if the note most recurrent in a phrase was C and the second and third most recurrent notes were G and E, we considered this phrase being in the tonality of C (the modality wasn't analyzed, because that can be read from the interval analyses presented before - see Fig. **??** and Fig. **??**).

The result of this tonality search is a macrostructure representing the tonal center of each phrase, akin to Schenker's Urlinie. The goal was to develop an algorithm capable of identifying the "skeleton" of the melody, following the principles of Schenkerian analysis. This result is depicted in Fig. 36, where each track (y-axis) shows the note values of the macrostructure (z-axis), and the x-axis represents the time length of each track. The corrisponding version in 2D can be seen in Fig. 37.

From this representation we can evaluate the changing of the structure and its variety, the direction that the given tracks have and if they have one. This is useful most of all to the determine the main differences between musical genres, whereas the rhythm and pitch analysis is more indicated when looking for stylistic characteristics (e.g. when looking for the differences between Chopin's melodies and Bach's).

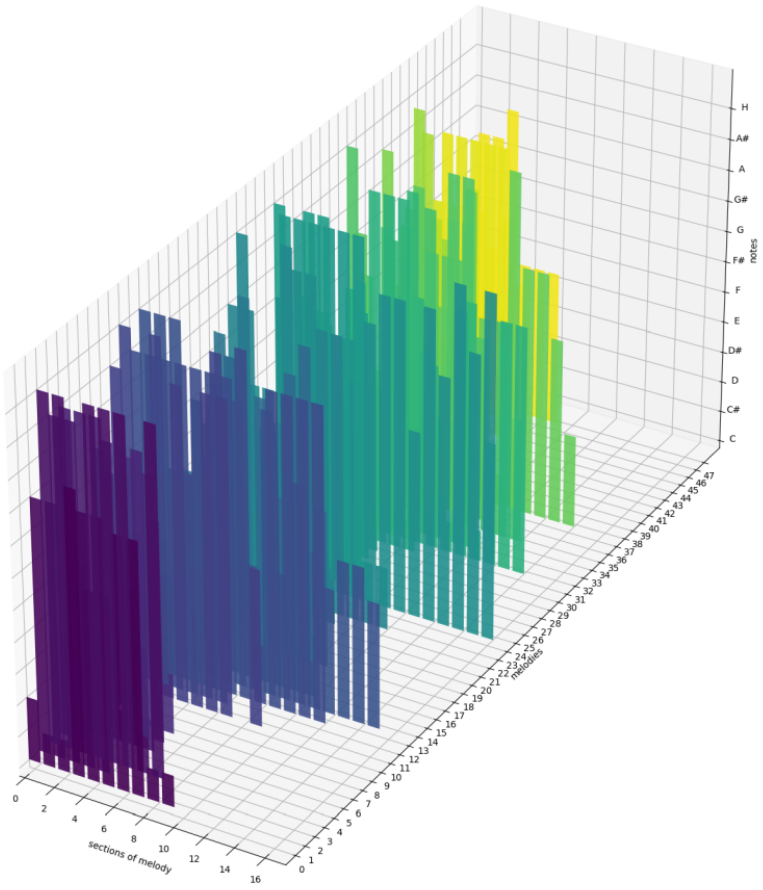


Fig. 36: Macrostructures of the dataset

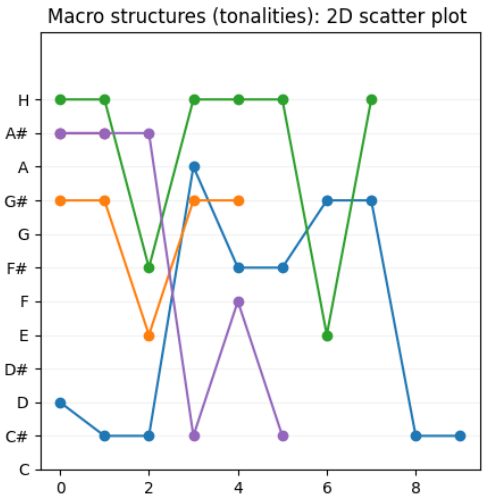


Fig. 37: Two-dimensional macrostructure's representation

6.3 "Synthesis" Tools

The second class of algorithms presented in this work utilizes analysis results and a given dataset to generate new melodies. Three approaches were tested: a generative genetic algorithm, probabilistic grammatical evolution, and a long-short term memory (LSTM) neural network.

The genetic algorithm rearranges segments of the dataset's melodies, making it useful for creating "musical collages" of different genres or styles that meet specific musical requirements defined by the fitness function.

The neural network was tested according to the literature presented in Chapter 4. The strength of neural networks lies in their potential to generate new combinations based on a trained model. LSTM is the algorithm most akin to the music composition process, as composers seek new solutions based on their knowledge and the genre they wish to compose in. However, a key challenge with neural networks is the limited ability to direct the creation process once the network is trained. Although we can't directly control the "composer" within the neural network, we can preprocess the training data or select acceptable results through supervised learning.

To address the neural network's limitations, a third approach was tested: probabilistic grammatical evolution. This method aims to closely monitor and control the generation process by providing not only a starting dataset but also a core structure to reflect.

6.3.1 Melody composition using Genetic Algorithm

The genetic algorithm was implemented according to the scheme showed in Fig. 38: initially, an initial population is generated. Subsequently, genetic operators are applied to produce offspring individuals, and finally, a selection process chooses the fittest individuals to serve as parents for the next generation.

Initial population and genetic operators

The initial population was made of MIDI sequences: a maximal length was chosen and every melody was divided in equal parts containing the same number of MIDI messages. From the MIDI messages only messages of type "note_on", "note_off" and "time_signature" were chosen.

At the end of this melody sectioning process we get melody segments of different sizes, implemented as list of arrays representing notes' pitches and lengths. The dynamics (the velocity parameter in MIDI messages) and the pedal changes were not considered.

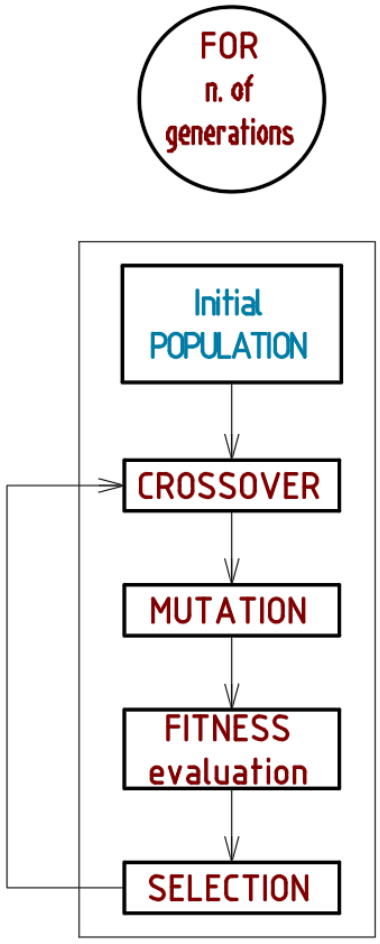


Fig. 38: Diagram of the Implemented Genetic Algorithm

For example the melody in Fig. 39

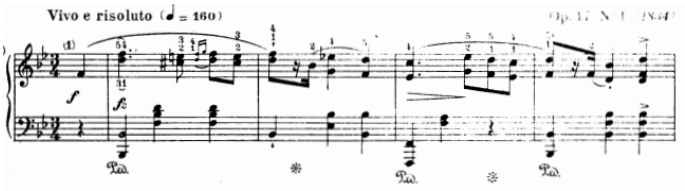


Fig. 39: First melody of the Chopin's mazurka op. 17 n. 1 (Chopin, 1834)

was converted into a list of arrays as is showed in Tab.2.

These segments (chromosomes) form the population for the given generation. Then, these chromosomes are variated using the "one point crossover" mechanism. According to a given probability, the crossover takes randomly two parents and chooses a random point. This point breaks the original chromosome melody segment into two pieces according to the index given by the random point value. In Fig.40 can be seen how it works if the point is the middle index of the melody vector.

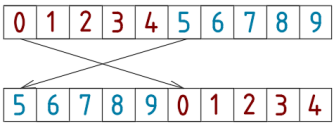


Fig. 40: Crossover scheme

When the crossover has created enough individuals, the mutation of some of these individuals takes place. During the mutation a random element from the melody is chosen and replaced by another random one.

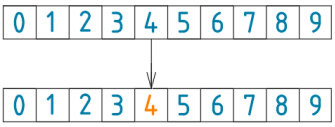


Fig. 41: Mutation scheme

After the individuals are generated the fitness function determines the best of them that will become the parents for the new generation.

The fitness function

The fitness function was designed to evaluate various musical characteristics desirable in a melody from the Classic or Romantic period. These characteristics are quantified using specific functions and then linearly combined according to their

Tab. 2: Example of the first segment of melody from Chopin's mazurka op. 17 n. 1

type	numerator	denominator		
['time_signature'	4	4	0]	

type	pitch	time	velocity	channel
['note_on'	66	28	63	0]
['note_on'	68	98	52	0]
['note_off'	66	18	64	0]
['note_on'	66	17	62	0]
['note_off'	68	21	64	0]
['note_on'	65	10	63	0]
['note_off'	66	2	64	0]
['note_off'	65	14	64	0]
['note_on'	66	0	70	0]
['note_on'	68	5	55	0]
['note_off'	66	3	64	0]
['note_off'	68	19	64	0]
['note_on'	66	97	55	0]
['note_on'	69	1	66	0]
['note_off'	66	12	64	0]
['note_off'	69	100	64	0]
['note_on'	66	15	61	0]
['note_on'	69	3	61	0]
['note_on'	71	92	55	0]
['note_off'	69	16	64	0]
['note_off'	66	14	64	0]
['note_on'	69	12	55	0]
['note_off'	71	19	64	0]

importance. The resulting fitness function has the following form:

$$F(f(x_m), f(x_r), f(x_{rv}), f(p_i), f(x_e)) = \\ = a\, f(x_m) + b\, f(x_r) + c\, f(x_{rv}) + d\, f(p_i) + e\, f(x_e) \quad (13)$$

Each of these functions, denoted as f, is designed such that their trend accurately represents how the given parameter should manifest in the melody. It is assumed that if their value exceeds 15, the parameter is sufficiently represented in the melody. Ideally, the optimal solution to the problem will then have a fitness evaluation equal to:

$$f^* = (a + b + c + d + e + g) \cdot 15 \quad (14)$$

where the coefficients a, b, c, d, e, g are chosen according to the user's musical preferences.

Melody patterns

This parameter calculates the number of repeating 3-notes patterns in the melody (it doesn't check the rhythm of these patterns). If the pattern is present in the melody at least two times, the fitness function is increased by one. That means that if the final number of repeating patterns is 10, there are 10 3-notes patterns that are present at least two times in the melody. This resulting number is then normalized according to the function:

$$f(x_m) = 15,7 \cdot e^{-\frac{(-3x_m+12)^2}{2^{9.8}}} \quad (15)$$

where x_m is the number of repeating patterns.

Rhythm patterns

Similarly to the melody patterns, also the number of 3-notes rhythmic patterns, that are repeated in the melody segment at least two times, is calculated. This number is then normalized according to the logarithmic function:

$$f(x_r) = 5 \cdot \log(5 \cdot x_r - 55) + 13 \quad (16)$$

where x_r is the number of repeating patterns.

Rhythmic variety

This parameter calculates the variety of note durations in a melody, the function used to normalize this number is the same as the function for calculating the rhythm patterns (Eq. 17), where x_{rv} will be the number of different rhythm types:

$$f(x_{rv}) = 5 \cdot \log(5 \cdot x_{rv} - 55) + 13 \quad (17)$$

Time lengths are rounded time values derived from MIDI messages. These numbers represent the number of ticks that occur between one message and another in a MIDI

file, specifically between a “note_on” message and the corresponding “note_off”ss message for the same note pitch. When MIDI files are sourced from recordings of real musical performances, notes that should be the same length (e.g., quarter notes according to the sheet music) often have varying numbers of MIDI ticks. This is because performers rarely play notes with identical durations, especially when interpretational rubato is involved. Additionally, MIDI ticks have microsecond precision, so any variation in length between identical rhythms will be noticeable. Therefore, after calculating the duration of a given note in MIDI ticks, the number is converted to the number of beats that the note lasts (see the notes' duration diagram in Fig. 24). This conversion helps to significantly reduce the sensitivity to variations in note lengths.

Interval variety

What makes a melody interesting is often also the intervals of which it is made. To check this parameter an interval polynom was defined, in which different intervals have different weights:

$$p_i = a\,\mathrm{V} + b\,\mathrm{IV} + c\,\mathrm{VII} + d\,\mathrm{II} + e\,\mathrm{III} \tag{18}$$

where the roman numbers indicate the number of times the specific musical interval has been detected in the melody segment at hand (V - perfect fifth, IV - perfect fourth, III - minor and major third, VII - minor and major seventh and II - minor and major second). Note that the intervals' weights, represented by the coefficients a, b, c, d, e can be set according to the desired output's characteristics. The number resulting from the interval polynom p is then normalized by the function:

$$f(p_i) = \frac{5 \cdot \log(5 \cdot p_i - 55)}{1800} \tag{19}$$

Ending

The last parameter that is being checked is the interval distance between the start note of the melody and the last one, because melodies usually end in the same range as the starting note.

$$f(x_e) = 10 \cdot \tan^{-1}(88 \cdot (x_e - 0,65)) \tag{20}$$

where x_e is the interval distance between the first and last note of the melody.

6.3.2 Melody composition using Probabilistic Grammar Evolution

The PSGE algorithm has three main components: the grammar definition, the grammar evolution algorithm and the genetic search engine. The pseudo code showing the structure of the applied algorithm is presented in Alg.1. First, a definition of a grammar is necessary. The grammar defined for this application reflects the aim that the user wants to reach, the form of the possible solutions (the melodies we want to create in this case).

Algorithm 1 PSGE

```
procedure MAIN(population_size, crossover_rate, mutation_rate,
generations, dataset, grammar)
    for gen in generations do
        for i in population_size do
            codon = random(0,1)
            selected_rule = EXPAND(symbol, codon, grammar, depth, max_depth)
            genotype = CREATE(genotype, symbol, grammar, depth, max_depth)
            genotypes.append(genotype)
        end for
        genotypes = crossover(genotypes, crossover_rate)
        genotypes = mutation(genotypes, mutation_rate)
        for gen in genotypes do
            phenotype = MAP(gen, positions_to_map, symbol, depth,
max_depth, grammar)
            phenotypes.append(phenotype)
        end for
        for phen in phenotypes do
            parsed = PARSE(phenotype)
            MIDI = CONVERT(parsed, pitch_range)
        end for
        Fitness =EVALUATE(MIDI, rules)
        Selected = SELECTION(fitness, rules, MIDI)
        Grammar = UPDATE(selected, learning_factor)
    end for
end procedure
```

A classical music grammar

The grammar used for this testing application reflects the structure of MIDI files, as the datasets given are made of files in MIDI format. Therefore, the start symbol of the grammar is:

$$< Start > \ ::= < Meter >< + >< Sequence > \tag{21}$$

The meter being the accentual structure of the melody, as described in chapter 6.1. Following the meter definition, the recursive sequence generation begins:

$$\begin{aligned} < Sequence > \ ::= & < Note >< + >< Sequence > \ | \\ & < Intervals >< zip >< Rhythms >< + >< Sequence > \end{aligned} \tag{22}$$

New elements are added to the sequence in two ways: either adding single pitches or appending an interval and rhythmic sequence of choice. The operator is a *zip* function that pairs every interval to a sequence of notes' durations (a rhythm). The resulting sequence of intervals and durations will be translated into pitches during the genotype to phenotype transformation. The pitches will be calculated from the pitches that precede. For example, if there is a G note and a sequence of [IImaj, V] intervals right after, the phenotype will be: $G - A - E$, because $G - A$ form a major second (IImaj) interval and $A - E$ form a perfect fifth (V) interval.

Adding a pitch ($< Note >$) means defining a note, its octave and its duration:

$$< Note > \ ::= < Tonality >< + >< Duration >< + >< Octave > \tag{23}$$

Notes aren't added to the sequence randomly; they reflect the tonality of the melodic segment. Certain notes are used more frequently than others because each note in a tonality has its own function and significance. For example, if a melody segment is in C major, the notes C and E form a tonic harmony, making them more frequent and important than notes like D or A, which primarily serve as passing or predominant harmonic notes in C major. Additionally, in C major, the note G is crucial. The G chord (comprising G, B, D, and F) functions as the dominant, creating tension and a sense of direction that is resolved only by the tonic chord.

Thus, the tonality is first chosen according to a predefined macrostructure or the statistical information derived from dataset analysis. Then, pitches are selected based on a preset probability distribution among scale degrees (see Tab. 3):

Note that the sum of the probabilities of each scale degree is not 1: that is due to the presence of chromaticisms in the choice of notes. Chromaticisms are made of notes outside the current tonality, they are used as melody embellishments, or to make harmonic modulations happen. The probabilities chosen in Tab. 3 are just for reference, they can be set freely according to the application and the user's

Tab. 3: Table of the tonal scale degrees

Scale degree	Name	Probability	Corresponding note in C major
I	Tonic	0.3	C
II	Supertonic	0.04	D
III	Mediant	0.15	E
IV	Subdominant	0.04	F
V	Dominant	0.15	G
VI	Submediant	0.04	A
VII	Leading tone	0.04	H
	$Total:$	0.76	

preferences. If the desire is to compose non-tonal music, higher probabilities will not be given to the scale degrees, but instead even probability values will be given to all the notes.

Lastly, the terminal symbols of this grammar are all the musical parameters that will be needed to compile a MIDI file: the meter values, the pitches, the tonalities, the intervals, the rhythmic sequences, the notes' durations values and the octaves. These values are either extracted from a given dataset or given by the user and can have any form needed. For example if the aim is the generation of sound, the pitches won't be MIDI numbers but frequencies, and so on.

On the other hand, the start symbol, operators and the non-terminals of this grammar are:

$$\begin{aligned} \text{Non Terminals} &::= < Sequence >, < Meter >, < Octave >, \\ &\quad < Note >, < Rhythms >, < Intervals > \\ &\quad < Tonalities >, < Degrees >, \\ &\quad < Durations > \\ \text{Axiom} &::= < Start > \\ \text{Operators} &::= < + >, < zip > \end{aligned} \tag{24}$$

Probabilistic context-free grammar

Probabilities are assigned to each rule to vary their outcomes according to the user's preferences or the parameters of the given dataset. If the starting point is a dataset and the goal is to generate melodies in a similar style, the probability values for each terminal are derived from the analysis tool's results (see section 6.2). These probability values are then incorporated into the grammar, with terminals defined based on the features extracted from the dataset during analysis:

1. The meters
2. The octave ranges
3. The time durations
4. The tonalities
5. The interval sequences
6. The rhythmic sequences

The analysis returns for each parameter the most used elements and their probabilities of occurrence and the algorithm takes these results and builds the grammar according to them automatically. For example, the grammar rule for interval sequences will be the defined as:

$$
\begin{aligned}
&< IntervalSequence >::= \\
&\quad < FirstInterval >, \quad \text{prob: first int. prob.} \\
&\quad < SecondInterval >, \quad \text{prob: second int. prob.} \\
&\quad < ThirdInterval >, \quad \text{prob: third int. prob.} \\
&\quad < OtherIntervals >, \quad \text{prob: other int. prob.}
\end{aligned}
\tag{25}
$$

All the other Non Terminal symbols will have a similar definition, except for the tonality symbol, which will be defined as follows:

$$
\begin{aligned}
&< Tonality >::= \\
&\quad < FirstTonality >< + >< Degree >, \quad \text{prob: first ton.} \\
&\quad < SecondTonality >< + >< Degree >, \quad \text{prob: second ton.} \\
&\quad < ThirdTonality >< + >< Degree >, \quad \text{prob: third ton.} \\
&\quad < OtherTonality >< + >< Degree >, \quad \text{prob: other ton.}
\end{aligned}
\tag{26}
$$

And the following *Degree* rule has the form based on the information explained in Tab. 3. The degree rules don't compile only the scale degrees but all 12 scale notes, because it considers the possibility of chromaticisms. The symbol names are the numeric values of the distance between the resulting pitch and the tonality (its tonic note). For example, the symbol $< 5 >$ means that the resulting note will be

5 semitones above the tonic (in case of C major tonality it will be the note F).

$$
\begin{array}{lllr}
< Degree >::= & <+0>, & \text{prob:} & \text{I prob.} \\
 & <+1>, & \text{prob:} & \text{II prob.} \\
 & <+2>, & \text{prob:} & \text{II prob.} \\
 & <+3>, & \text{prob:} & \text{III prob.} \\
 & <+4>, & \text{prob:} & \text{III prob.} \\
 & <+5>, & \text{prob:} & \text{IV prob.} \\
 & <+6>, & \text{prob:} & \text{IV+ prob.} \\
 & <+7>, & \text{prob:} & \text{V prob.} \\
 & <+8>, & \text{prob:} & \text{VI prob.} \\
 & <+9>, & \text{prob:} & \text{VI prob.} \\
 & <+10>, & \text{prob:} & \text{VII prob.} \\
 & <+11>, & \text{prob:} & \text{VII prob.}
\end{array}
\tag{27}
$$

Grammatical Evolution

Once the grammar is defined, it can be implemented in the grammatical evolution algorithm. First it is necessary to expand the grammar symbols, this is made by the *generate expansion* procedure (Alg.2 (Mégane, 2022)):

When the star symbol is expanded, the process of creating new individuals begins (Alg.3, (Mégane, 2022)). During this process the genotype is generated by appending every expanded symbol and its probability value. This procedure is recursive: when a new symbol is added to the genotype, a new expansion with a new codon value takes place, resulting in new symbols to create individuals from. This process is continued until the symbol obtained is a terminal symbol.

The genotypes obtained from the create procedure are then taken by the genetic functions of crossover and mutation. They are recombined during crossover according to the set crossover rate and then some of the elements are mutated according to the mutation rate (to know how the crossover and mutation works see section 6.3.1).

After the genotypes are genetically processed, the genotype to phenotype mapping takes place (Alg. 4):

Algorithm 2 Generate expansion

```
procedure EXPAND(symbol, codon, grammar, depth, max_depth)   ▷ Expand grammar symbols according to codon's value
    if depth < max_depth then
        for element in grammar[symbol] do
            cum_prob += element(prob)
            if codon ≤ cum_prob then                          ▷ Check the probability value
                selected_rule = element(rule)
                return selected_rule                          ▷ Return symbol's expansion rule
            end if
        end for
    else
        rules = get_non_recursive_rules(grammar[symbol])      ▷ Function to get non recursive rules
        total_prob = sum(element(prob) for element in rules)
        for element in rules do
            cum_prob += element(prob)/total_prob
            if codon ≤ cum_prob then:
                selected_rule = element(rule)
                return selected_rule                          ▷ Return symbol's expansion rule
            end if
        end for
    end if
end procedure
```

Algorithm 3 Create individual

procedure CREATE(*symbol*, *genotype*, *grammar*, *depth*, *max_depth*) ▷ Create an individual's genotype
    $codon = \text{random}(0, 1)$
    $genotype(symbol).\text{append}(codon)$
    $selected\_rule = \text{expand}(symbol, codon, grammar, depth, max\_depth)$
    $rules.\text{append}(selected\_rule)$
    $expansion\_symbols = grammar(symbol, rule)$
    for *symbol* in *expansion_symbols* **do**
        if not is_terminal(*symbol*) **then** ▷ Check if symbol is terminal
            create(*symbol*, *codon*, *grammar*, *depth*, *max_depth*)
        end if
    end for
    return *genotype* ▷ Return generated genotype
end procedure

Algorithm 4 Genotype to phenotype mapping

procedure MAP(*genotype*, *symbol*, *positions_to_map*, *grammar*, *depth*, *max_depth*) ▷ Get the phenotypes
    $position = positions\_to\_map(symbol)$
    if $position \geq \text{len}(genotype(symbol))$ **then**
        $codon = \text{random}(0, 1)$
    else
        $codon = genotype(symbol, position)$
    end if
    $selected\_rule = \text{expand}(symbol, codon, grammar, depth, max\_depth)$
    $rules.\text{append}(selected\_rule)$
    $expansion\_symbols = grammar(symbol, rule)$
    $positions\_to\_map(symbol) += 1$
    for *symbol* in *expansion_symbols* **do**
        if not is_terminal(*symbol*) **then** ▷ Check if symbol is terminal
            $phenotype += symbol$
        else
            $phenotype += \text{map}(genotype, symbol, positions\_to\_map, grammar, depth, max\_depth)$
        end if
    end for
    return *phenotype* ▷ Return generated phenotype
end procedure

The procedure of mapping is applied to each genotype until a phenotype representing the whole melody is created. At this point the phenotype is translated into a MIDI format musical file, that can be evaluated by a fitness function. Based on the results of the fitness evaluation the probabilities of the grammar rules are updated to get better ("more fit") melodies. The rules update procedure is showed in Alg. 5.

Algorithm 5 Rules update

procedure UPDATE($grammar, selected\_rules$, $learning\_factor$) ▷ Update the probability values in the grammar
  for $recurrence$ in $selected\_rules(recurrence)$ **do**
    $tot\_prob += recurrence$
  end for
  for $rule$ in $selected\_rules$ **do**
    $rule\_use = rule(recurrence)$
    if $rule\_use > 0$ **then**
      for i in $grammar(rule)$ **do**
        if i in $rule$ **then**
          $new\_prob = \min(prob + learning\_factor \cdot rule\_use/tot\_prob)$
          $grammar(i, prob) = new\_prob$
        else
          $new\_prob = prob - learning\_factor \cdot prob$
          $grammar(i, prob) = new\_prob$
        end if
      end for
    end if
  end for
  return $grammar$ ▷ Return updated grammar rules
end procedure

The learning factor updates the rules in two different ways depending on the fact if the rule was used in the mapping or not (Mégane, 2022). If the rule was used, the following equation is used:

$$prob_i = \min(prob_i + \lambda \cdot \frac{rec_i}{\sum_k rec_k}, 1) \tag{28}$$

where rec_i is the number of occurrences of the given rule in the mapping and the $\sum_k rec_k$ is thentotal sum of all occurrences. On the contrary, if the recurrence number is zero, the probability is updated according to the equation:

$$prob_i = prob_i - \lambda \cdot prob_i \tag{29}$$

Genetic operations: crossover, mutation and selection

Mutation and crossover are applied to the genotype before mapping, that means that just the codons are modified. During crossover a segmentation of the genotypes and a recombination of its parts is realized:

Before crossover		
	Parent 1	Parent 2
$< Meter >$	[0.27]	[0.53]
$< Sequence >$	[0.14, 0.32, 0.77]	[0.45, 0.22, 0.67]
$< Duration >$	[0.65, 0.18]	[0.16, 0.11]
$< Tonality >$	[0.93, 0.42]	[0.39, 0.02]

(30)

After crossover	
	Offspring
$< Meter >$	[0.27]
$< Sequence >$	[0.14, 0.32, 0.77]
$< Duration >$	[0.16, 0.11]
$< Tonality >$	[0.39, 0.02]

Mutation changes randomly some codons of the genome, according to a mutation rate that is preset by the user:

Before mutation	
$< Meter >$	[0.27]
$< Sequence >$	[0.14, 0.32, 0.77]
$< Duration >$	[0.16, 0.11]
$< Tonality >$	[0.39, 0.02]

(31)

After mutation	
$< Meter >$	[0.88]
$< Sequence >$	[0.14, 0.32, 0.77]
$< Duration >$	[0.16, 0.53
$< Tonality >$	[0.39, 0.02]

The selection of the fittest individual is made after the genotype to phenotype mapping and the translation of the phenotype into a MIDI file. That means that selection could be made in different ways, one of which could be a subjective selection process made by the author. In this work different variants of fitness functions were used.

Fitness function forms

The fitness function can have any form depending on what the aim of the user and the musical genre in question is. In this case two main fitness function forms were used a combination of non-linear functions and a full-linear version.

1. Linear combination of non-linear functions, with reference value of 15.
2. Linear combination of linear functions

The base form of the fitness function for every variant tested is the following:

```
fitness value = fast  + repeating_rhythms + rhythm_density +
 interval_variety + ending + mean_distance +
 high_low + rhythm_min + accents + note_variety
```

In the case of the use of non-linear functions, exponential, goniometric, logarithmic, and polynomial functions in order to get a value higher than 15 if the tested parameter is satisfied in the melody, as was done for the genetic algorithm (section 6.3.1). The functions used in this case are variants of the functions presented in section 6.3.1 (see Fig. 42).

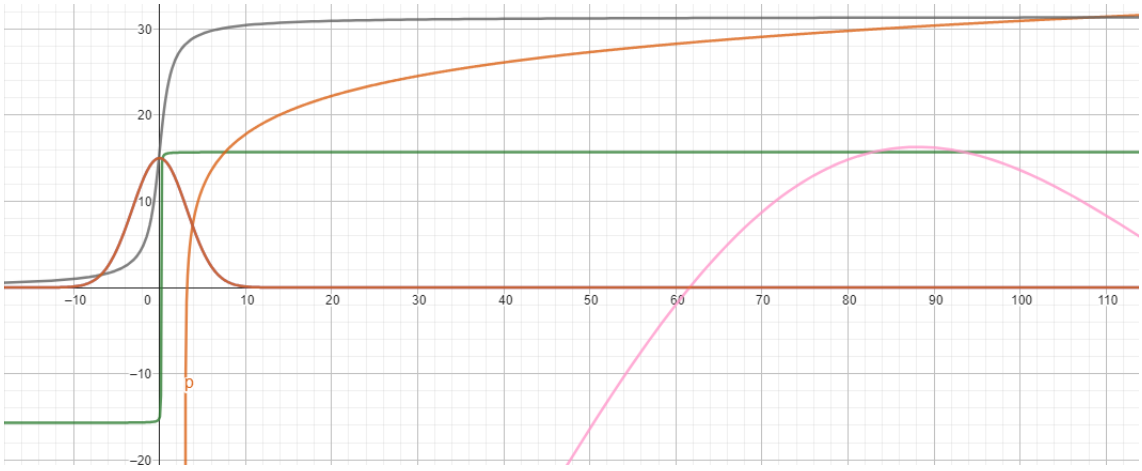


Fig. 42: Plots of some of the functions used in fitness evaluation

The variant that solely employs normalized fitness functions is intended to be more responsive to alterations. In this scenario, each parameter assessed by the fitness function is evaluated linearly: the coefficient is +1 if the parameter represents a desired characteristic of the melody, and -1 otherwise. A change of one unit in the given parameter reflects the minimal significant increase or decrease of that parameter.

For instance, to compute the parameter "repeating_rhythms," the density of each rhythmic value is initially calculated. This value is then normalized to change

by one unit each time the value varies by 0.1 (the designated sensitivity value), as follows:

$$\mathrm{r}_{norm}(x) = \begin{cases} 0 & \text{if} \quad x < 0,25 \\ \frac{x-0,25}{0,1} & \text{if} \quad 0,25 \leq x \leq 0,5 \\ 0 & \text{if} \quad x > 0,5 \end{cases} \tag{32}$$

6.3.3 Melody composition using LSTM

A long-short term memory (LSTM) neural network was evaluated using the same datasets employed for testing the genetic algorithm and grammatical evolution. The MIDI files were read and parsed using the Music21 library, which separates the data into notes and chords. Each note object includes its offset and octave during parsing, where the offset indicates the note's position in the piece. In the LSTM implementation utilized (based on Skúli, 2017), the arrays of notes and chords were then converted into arrays of integers. Each note is represented by a number based on its position in the sorted set of all notes used in the dataset.

Once the arrays of numbers are obtained from the dataset, input sequences of numbers are generated. The network output for each input sequence will be a note or chord that follows the input notes sequence. Various sequence lengths were tested.

The Neural Net model used was built using the Keras library. It is a sequential model, which used the following layers (Skúli, 2017):

1. Input layer: a LSTM layer with given input shape.
2. LSTM layers
3. Dropout layers: used to drop a fraction of input units.
4. Dense layers

Sequential models consist of layers, with each layer having one input tensor and one output tensor. The initial layer utilized is an LSTM layer with 512 units, employing a hyperbolic tangent function as the activation function. The recurrent state dropout is configured to 0,3.

Subsequent to the first layer, two additional LSTM layers with 512 units each are incorporated. Following these layers, a dropout layer with a dropout rate of 0,3 is added, which randomly sets input units to 0 based on the specified frequency rate. This dropout layer is employed to mitigate overfitting.

After the dropout layer, a regular densely-connected layer is introduced, implementing a rectified linear activation function ($a(x) = \max(x, 0)$). Following this layer, another dropout layer is applied. Finally, a Dense layer with the softmax function as the activation is added. The softmax function transforms a vector of real numbers into a probability distribution (Keras, 2024).

Once the model is trained, the output is generated by converting the normalized integer outputs into MIDI arrays, which can subsequently be played and evaluated. Based on how the model was created and how the dataset given was processed, only short melodies can be obtained, without directly controlling their overall structure (Jędrzejewska, 2018).

7 Application example: the Mazurkas' dataset

The analysis and composition methods underwent testing using various freely available MIDI datasets from the internet. This chapter presents the results obtained specifically from the MIDI dataset of Chopin's mazurkas (Midiworld, 2024).

7.1 Analysis and extracted features

The developed analysis application returns graphs and values to visualize the characteristics of the MIDI files dataset given. For the Chopin dataset the results can be seen in Fig. 43.

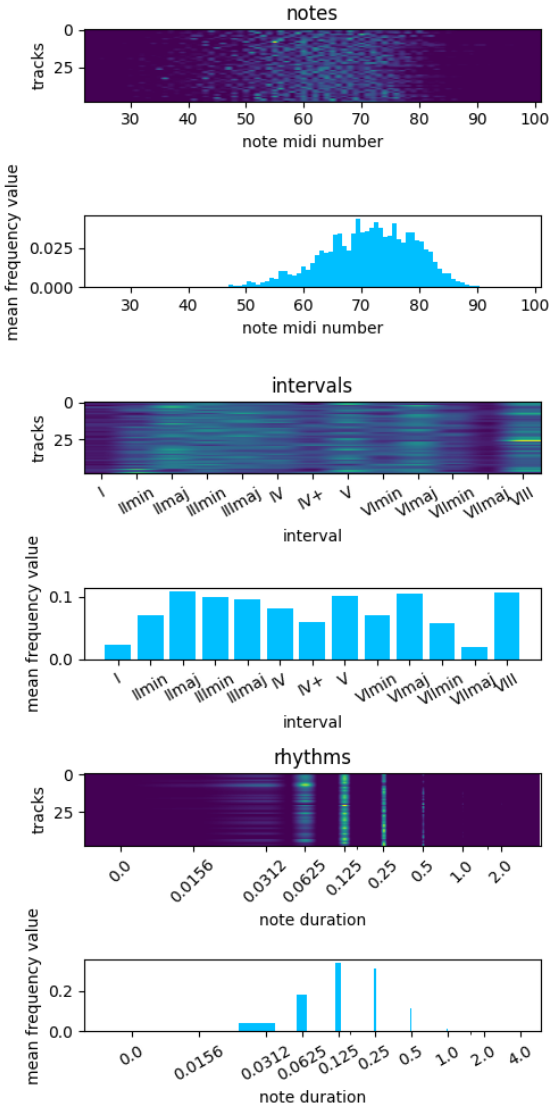


Fig. 43: Extracted features' distributions from the Chopin's mazurkas dataset

In the first two plots, it's evident that notes are distributed quite evenly within the range of 55 to 75 (G3 - E5). Additionally, diatonic intervals emerge as the most prevalent in all mazurkas: the most frequently occurring notes are typically spaced two MIDI numbers apart, as illustrated in detail in Fig. 44.

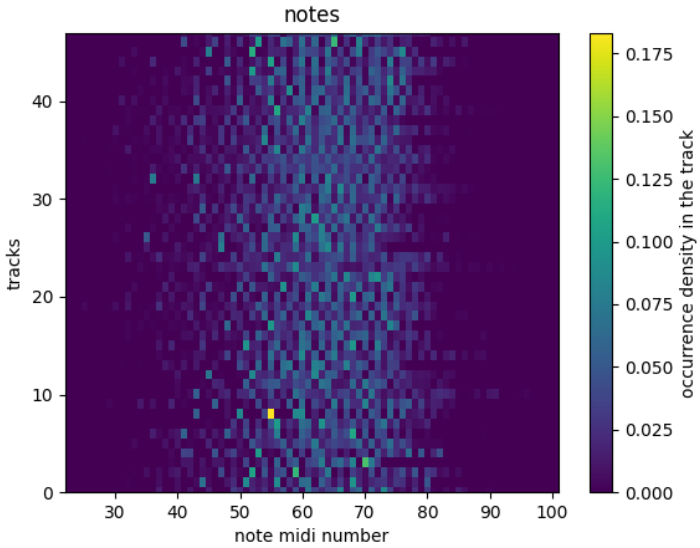


Fig. 44: Detail of the notes distribution over the Mazurkas' dataset

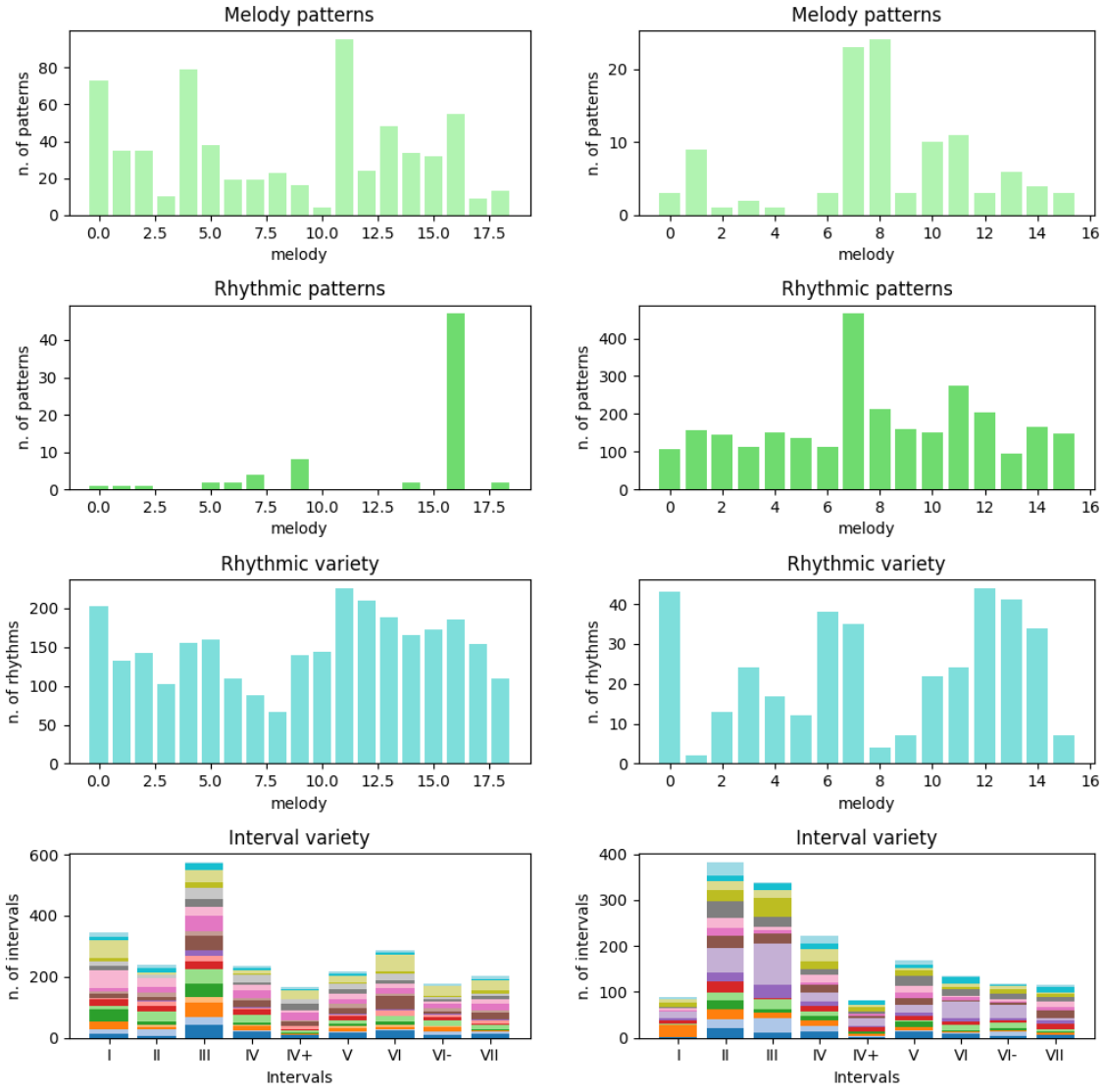
The "intervals" plots confirm that diatonic semitones (corresponding to intervals of IImaj in the graphs) are much more common than chromatic ones (IImin). Additionally, the intervals heatmap reveals that in most mazurkas, the most frequently used intervals are consistently II major, VII, III, V, and VI major.

Lastly, the "rhythms" plots indicate that the most frequently used note durations across all mazurkas are octaves, quarter notes, and sixteenths.

The second set of plots that the analysis tool examines melodic and rhythmic patterns of the melodies and the intervals most used overall. This set of plots can be used to compare different musical styles. As an example, a dataset of Bach's fugues was added to the mazurkas and the resulting histograms are showed in Fig. 45. The Bach's fugues MIDI files were taken from (Bachcentral, 2018).

In these histograms, the first 15 tracks are Bach's fugues, while the remaining ones are Chopin's mazurkas. It can be observed that Chopin employs more melodical patterns, at the same time the mazurkas exhibit greater variability in note durations. Chopin's compositions demonstrate a broader rhythmic variety compared to Bach's fugues. Bach, in his fugues, tends to use a more monotonous rhythmic outline, with each track featuring only a few types of note durations.

Finally, the most commonly used intervals by both composers are similar, except for the interval of the II. There is a correlation between the low rhythmic

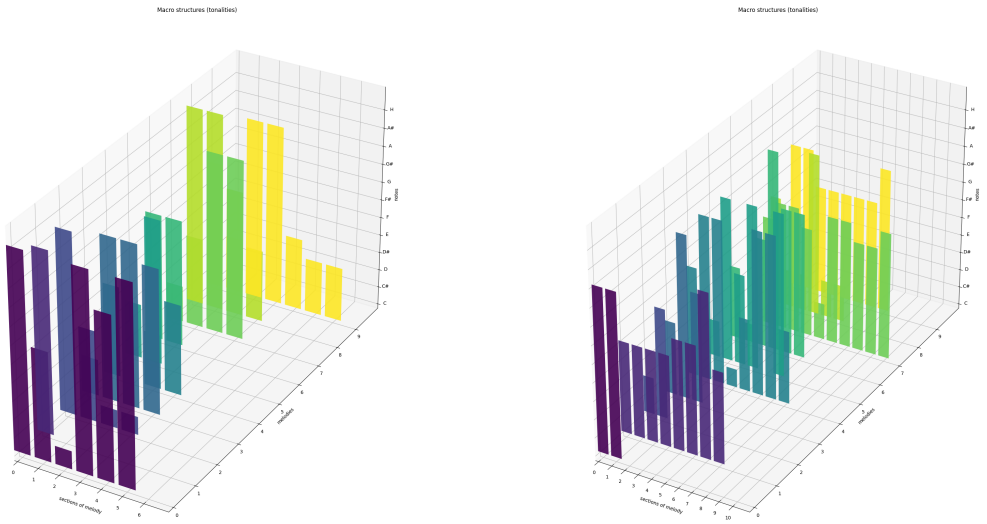


(a) (b)

Fig. 45: Extracted musical characteristics from Chopin's mazurkas (a) and Bach's fugues (WTC) (b)

variety in Bach's compositions and the high frequency of the II interval, indicating a significant presence of scale movements.

The third set of results of the "Analyst tool" consists of the plotted macrostructures. In Fig. 46, the macrostructures of 10 Chopin mazurkas are displayed alongside those of 10 Abba songs for comparison, taken from (Midiworld, 2024). The struc-



(a) (b)

Fig. 46: Macrostructures of a) Chopin's mazurkas and b) songs by Abba

tures of Abba's songs are more stable around a single tonal center, while Chopin's are shorter and more varied. As shown in the 2D representation (Fig. 47), the structures tend to conclude on the same notes they began with, or within the same tonal space, typically a III or V interval above.

7.2 Context-free grammar of the mazurkas' dataset

The second tool developed writes the grammar rules for the grammar evolution algorithm based on the fundamental characteristics of the dataset given. In the case of Chopin's mazurkas, the grammar has the following form:

```
 <start>:[{'exp': ['<meter>', '<seq>'], 'prob': 1.0}],
<meter>:[{'exp': ['<[array([3, 4])]>'], 'prob': 1.0}],
 <seq>:[{'exp': ['<note>', '<seq>'], 'prob': 0.4},
   {'exp': ['<int_seq>', '<op>', '<rhy_seq>', '<seq>'], 'prob': 0.6}],
 <op>:[{'exp': ['<zip>'], 'prob': 1}],
```

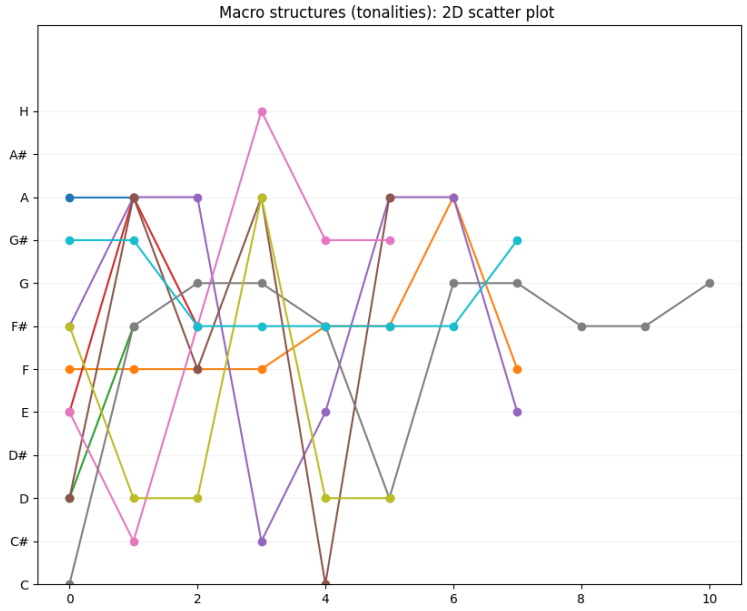


Fig. 47: Detail of the macrostructures of Mazurkas' dataset

```
<note>:[{'exp': ['<ton>', '<time>', '<octave>'], 'prob': 1.0}],
<octave>:[{'exp': ['<60>'], 'prob': 0.637},
  {'exp': ['<72>'], 'prob': 0.348}, {'exp': ['<84>'], 'prob': 0.0133}],
<time>:[{'exp': ['<8>'], 'prob': 0.421},
  {'exp': ['<16>'], 'prob': 0.267},
  {'exp': ['<4>'], 'prob': 0.184}, ...],
<ton>:[{'exp': ['<C#>', '<degree>'], 'prob': 0.166},
  {'exp': ['<D>', '<degree>'], 'prob': 0.152}, ...],
<degree>:[{'exp': ['<+0>'], 'prob': 0.3},
  {'exp': ['<+1>'], 'prob': 0.0399},
  {'exp': ['<+2>'], 'prob': 0.0399}, ...],
<int_seq>:[{'exp': ['<[ 2 -2 -1  1]>'], 'prob': 0.000584},
  {'exp': ['<[ 2 -6 -1  5]>'], 'prob': 0.00116}, ...],
<rhy_seq>:[{'exp': ['<[32  4  8  2]>'], 'prob': 0.00296},
  {'exp': ['<[ 8 16 32  4]>'], 'prob': 0.0151}, ...],
```

Note that it is possible for the author to implement other rules based on the desired results, some of the rules can be eliminated, or changed. For example, the macrostructure of the created melody can be set as follows, given the macro structure variable:

```
self.macro = [‘C’, ‘G’, ‘C’]
```

the grammar rule <ton> will then be modified accordingly:

```
if len(self.macro)>0:
   ’<ton>’ = [{"exp" : ["<macro>","<degree>"], "prob": 1.0}]
```

The probabilities values in the grammar are the results of the feature extractions, as well as some of the terminals of the rules, as for example the rhytmic and interval patterns. In this case pattern lengths of 4 elements were presented, but the algorithm works with any length needed. Patterns of three and four notes were tested.

7.3 Grammatical evolution algorithm

The grammatical evolution algorithm was implemented according to the description in section 10. Different ways of building the new population at every generation iteration were tested, as long as different fitness function forms:

Random tree initialization
Individuals are generated by randomly building derivation trees up to a specified depth limit (Ortega, 2007). Therefore, only information about the updated grammar's probabilities is passed from the older generation to the new one. In this case the best fitness function values throughout the generations don't have a positive gradient (see Fig.48), but the results from a musical point of view aren't disappointing (considering the melodies which got the highest fitness values during training), as can be seen in Fig. 49. As can be deduced from the melody profile presented in

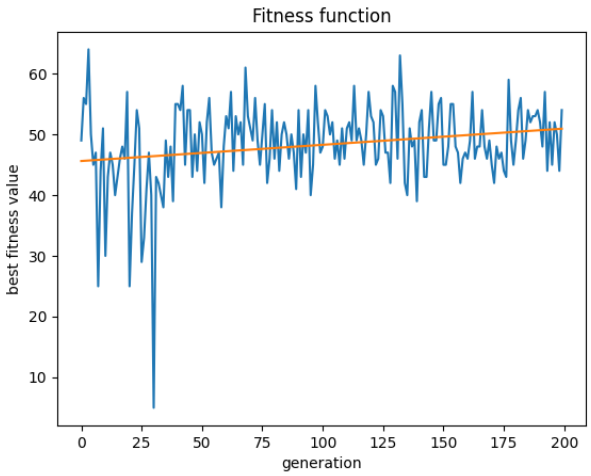


Fig. 48: Fitness function when using the random tree initialization

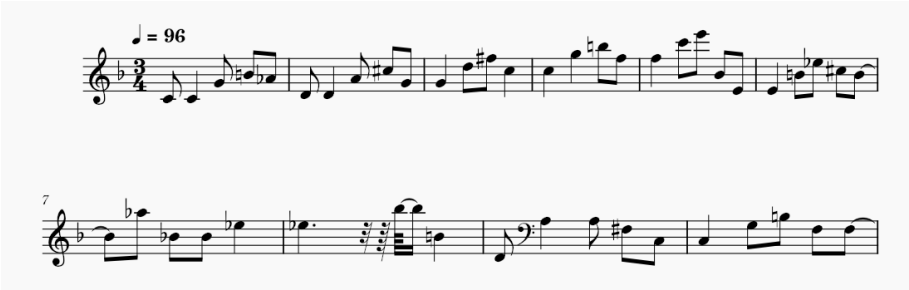


Fig. 49: Generated melody example

Fig.49, the fitness function was selected in order to get repeating rhythmic patterns and a narrow pitch range.

Truncation selection

The second selection mechanism chosen was the truncation selection: in this case, the best individuals from the previous generation replace the worst ones of the new generation. The fitness function values for 100 generations are showed in Fig. 50. The best melody obtained in the last generation is showed in Fig. 51

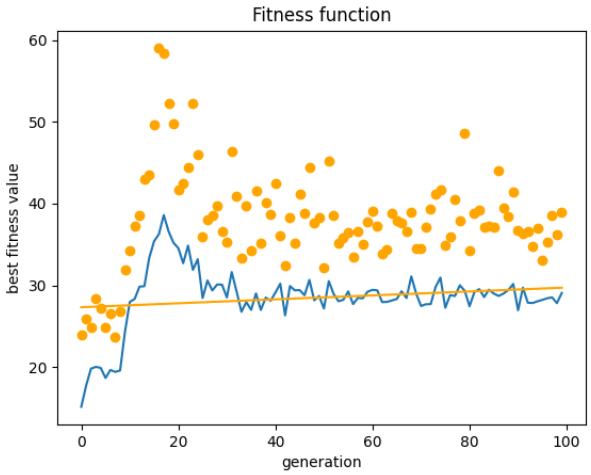


Fig. 50: Fitness function when using truncation selection

Fitness function with non-linear components

The “mean_distance” value of the fitness function is zero unless the mean distance between notes in the melody has a maximum value of 5 MIDI notes, according to the exponential function in Fig.52. The same function is used also for the values of the parameter “high_low”, which rewards those melodies, in which the distance between the highest pitch and the lowest is lesser than an octave.

An example of fitness values during training is presented in Fig. 53.

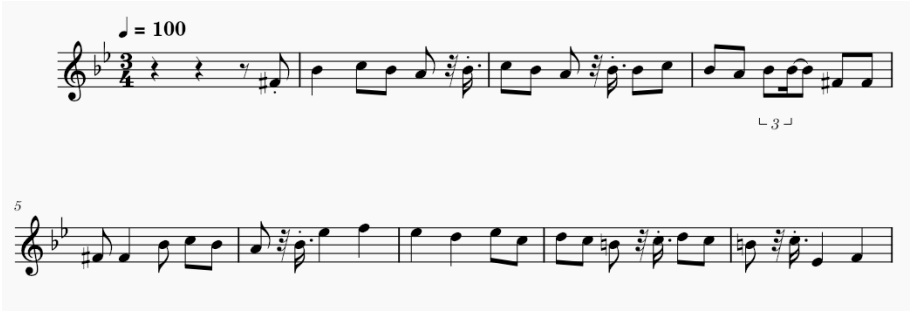


Fig. 51: One of the best melodies obtained using truncation selection

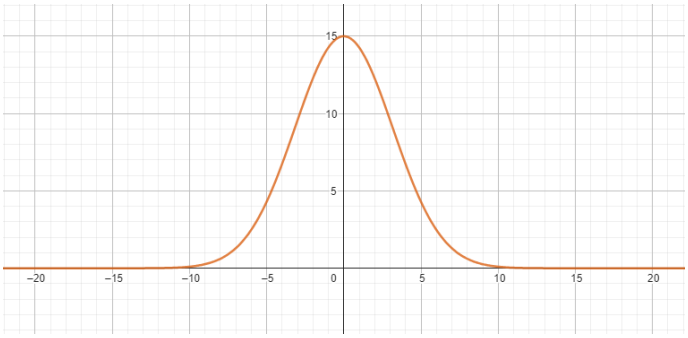


Fig. 52: Function used to evaluate some of the fitness function's parameters

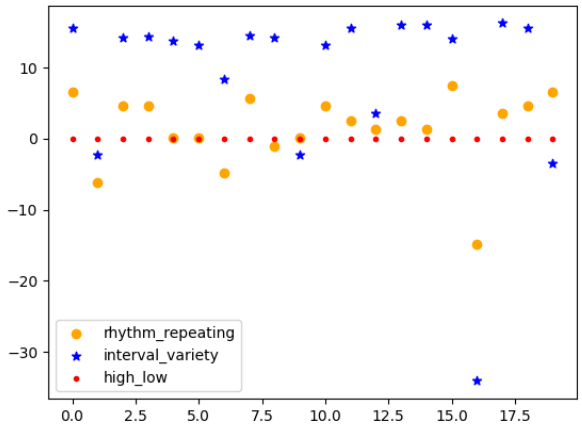


Fig. 53: Some of the fitness function parameters values during training

Linear fitness functions

In the case of using just linear functions, it can be possible to prioritize some parameters over others by adding coefficients to the fitness functions. For example, adding two coefficients:

```
fitness value = fast  + repeating_rhythms + rhythm_density +
 interval_variety + ending +  2 * mean_distance +
 2 * high_low + rhythm_min + accents + note_variety
```

The result with the highest fitness values got from this implementation is a rather rhythmically and melodically uniform melody (Fig. 54).

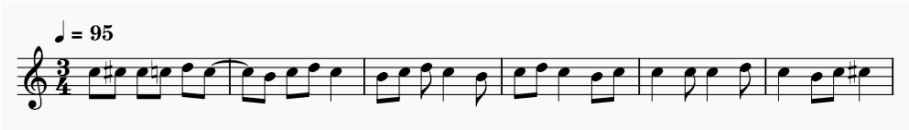


Fig. 54: Uniform melody when fitness function changed

7.4 Comparison with other algorithms

The same dataset was used with the genetic algorithm and the LSTM neural network.

Genetic algorithm

As in the case of the probabilistic grammar evolution algorithm, we considered only short sequences of notes with a pitch higher then F4 (65 in MIDI pitch number), the typical range for accompanied piano melodies in the romantic era.

For testing the genetic algorithm the fitness function presented in par. 6.3.1 was used, whose maximum values obtained in every generation are represented in Fig. 55.

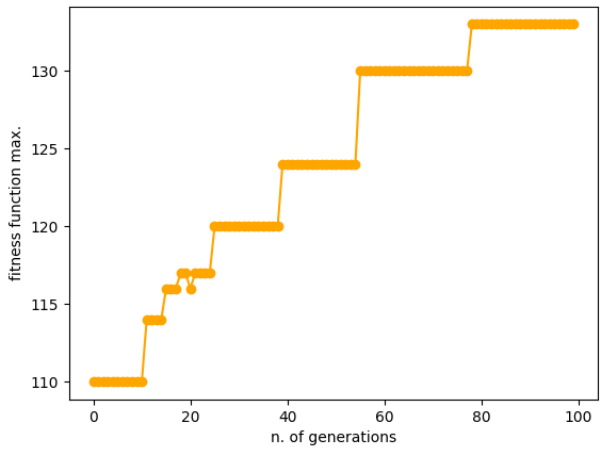


Fig. 55: Fitness function values for the genetic algorithm

An example of the melodies created in this case can be seen in Fig. 56.

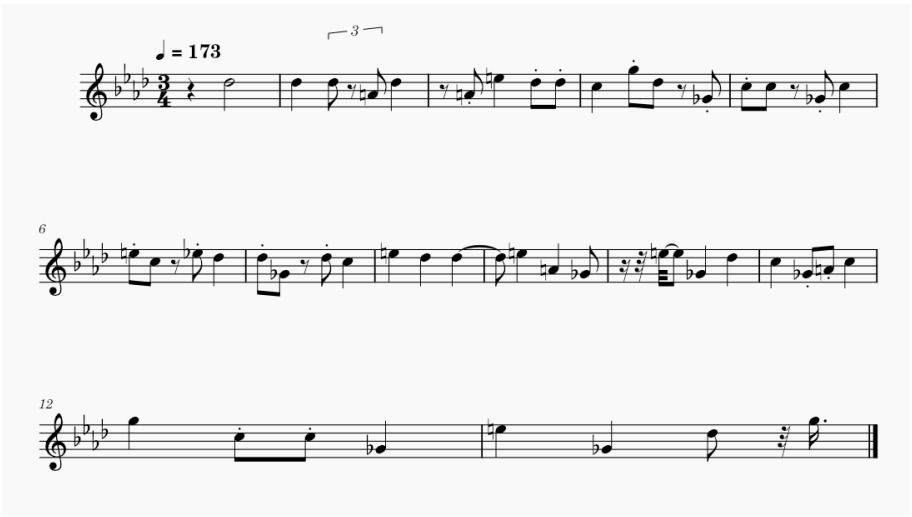


Fig. 56: Melody created by the genetic algorithm after 100 generations

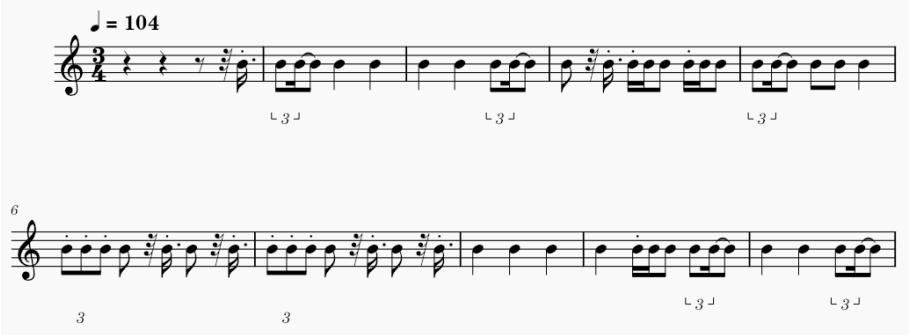


Fig. 57: Melody created by the genetic algorithm after 100 generations

When applying the fitness function used for the PSGE algorithm, the genetic algorithm has as result rhythmical figures without melody (Fig. 57). In case a fitness function that rewards only the intervals of third and fifth is used, the melodies created have a similar form as the excerpt in Fig. 58.

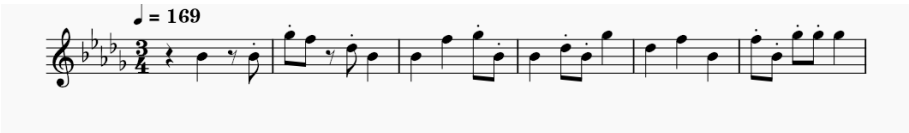


Fig. 58: Melody created with fitness function preferring III and V intervals

Long Short Term Memory network

Besides the genetic algorithm, the LSTM was tested, using the model presented in (Skúli, 2019). First it was tested with 100 epochs using the complete Mazurkas' dataset: the melodies and the left hand accompainment's chords. An excerpt of the result can be seen in Fig. 59. When using only notes with a pitch above F4

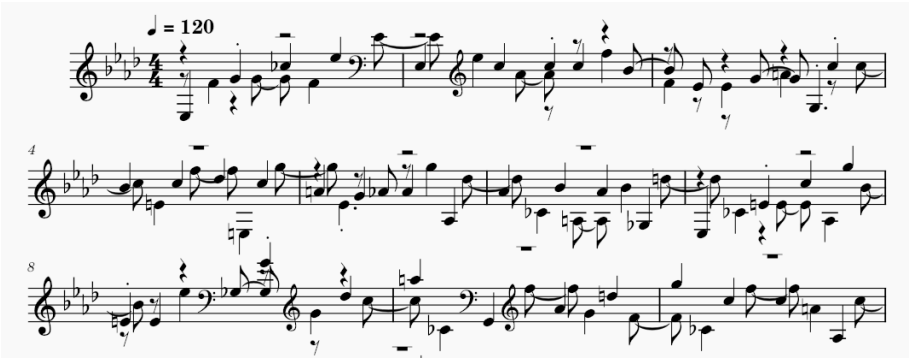


Fig. 59: Melody created by LSTM using dataset with accompaniment (chords)

(corresponding to MIDI number 65) and not considering the chords, the resulting

melodies are similar to the one presented in Fig. 60. The loss weight values trend is plotted in Fig. 61.

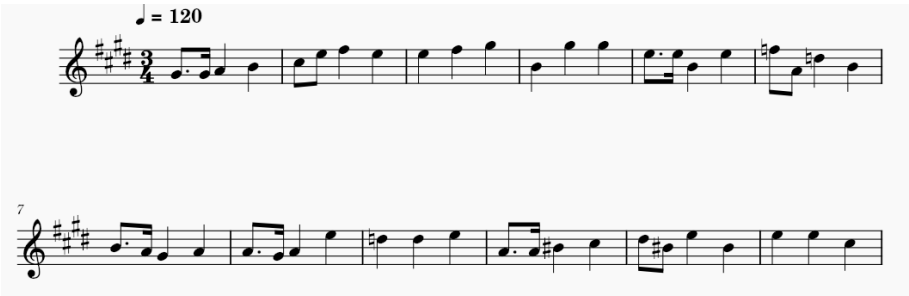


Fig. 60: Melody created by LSTM using one-voice dataset

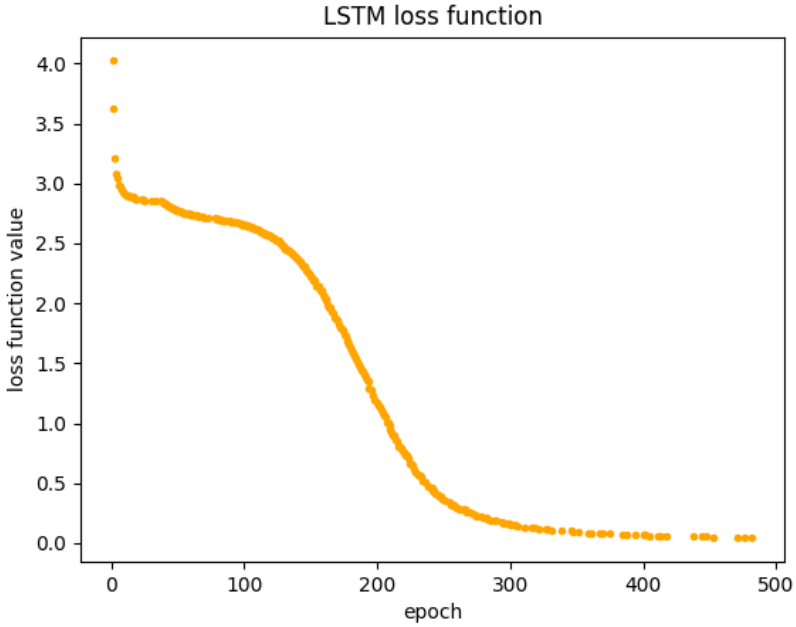


Fig. 61: LSTM loss function

8 DISCUSSION

The analysis tool can visualize the primary differences between musical genres or styles. As demonstrated in Fig. 45, the tool highlights key distinctions between Chopin's short pieces and Bach's style. The comparison conducted between Chopin's mazurkas and Bach's Well-Tempered Clavier is particularly relevant as Bach's work greatly inspired Chopin: he studied Bach's preludes and fugues extensively, forming the foundation of his musical education and teaching (Rosen, 1995). The histograms in Fig. 45 illustrate the contrasting approaches to melodic treatment, from Baroque homogeneity to rich Romantic textures. As musicologist Charles Rosen pointed out, Chopin disrupted the musical scene of his time with his monophonic pages, demonstrating the ability to minimize polyphony while still creating pieces with a clear harmonic structure (see the sonata's Finale in Fig. 20). This skill of Chopin's was directly inspired by Bach's counterpoint. Furthermore, the ability to harmonically treat a monophonic line without accompaniment is considered the best way to evaluate a composer's counterpoint abilities (Rosen, 1995).

The aim of the second set of codes presented than, the generation of monophonic melodies, can be considered an appropriate starting point for AMC developement.

To build the grammar of the probabilistic grammatical evolution algorithm some of the results of the "analyst tool" are used, this way the resulting musical style of the melodies obtained by this algorithm is closely controlled. From generation to generation the rules are updated based on the evolutionary search, making the resulting melodies gradually gain the desired form. After initialization, the melodies are controlled by the fitness function, which can be defined according to the author's preferences. Various versions of grammatical evolution were tested. When genotypes are generated at each iteration using the random tree initialization algorithm, interesting results are obtained, but there is no significant growth in fitness function values. The generated melodies often feature several rhythmic patterns and predominantly use intervals of fifths, fourths, and thirds. After a few hundred generations, a random number of melodies with high fitness function evaluations are produced, but the search mechanism to increase the number of these "better" melodies needs further exploration. The hypothesis is that this depends on the fitness function's structure.

Therefore, different types of fitness functions were tested. The first type (the "fitness function with non-linear components" described in Section 7.3) involves a mathematical evaluation of the most basic musical characteristics familiar to listeners of classical music melodies. The second type of fitness function focuses on melody characteristics directly influenced by the grammar: it does not evaluate the

overall structure of the melody or the number of repeating melodic or rhythmic patterns as in the first case. Instead, it assesses the intervals used, the mean distance between intervals, the spread of the melody across octaves, and accentuation. For example, in the case of mazurkas, the aim is to prioritize rhythmic patterns typical of mazurka dances (Fig. 62).

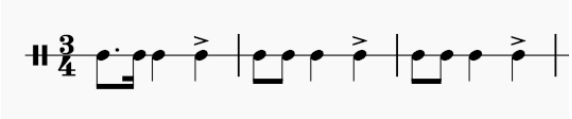


Fig. 62: Mazurka's rhythmical pattern

The results demonstrate that the fitness function is highly effective for generating a melody with specific characteristics, particularly when these characteristics pertain to the individual elements or shorter note patterns rather than the melody's macrostructure.

For instance, when the fitness function rewards a melody for having the accentuation typical of a mazurka dance, we achieve the melody shown in Fig. 63 after 16 generations. The progress of the fitness function is illustrated in Fig. 64, where the orange dots represent the highest fitness values achieved, and the blue line indicates the mean fitness value for each generation.

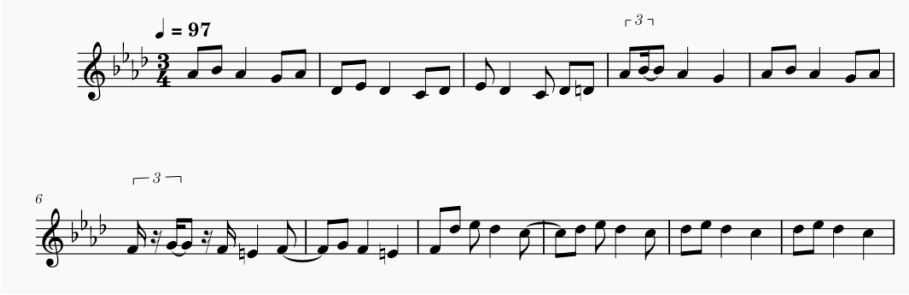


Fig. 63: Melody created by PGE with the mazurka's rhythm pattern

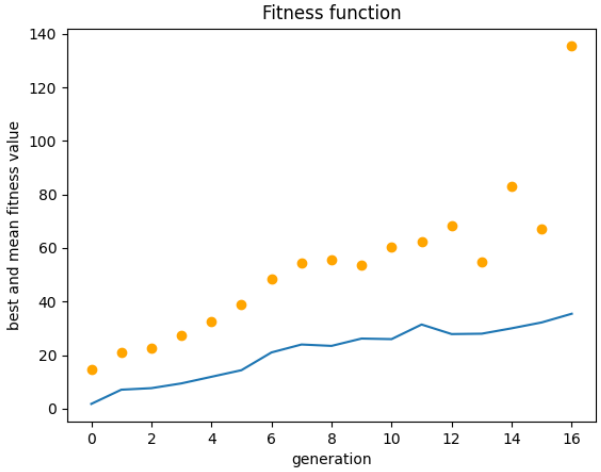


Fig. 64: Modified fitness function's progress

When the macrostructure [C, G, C] is given as input parameter, the melodies created reflect this structure (Fig. 65). Nevertheless this way of controlling the

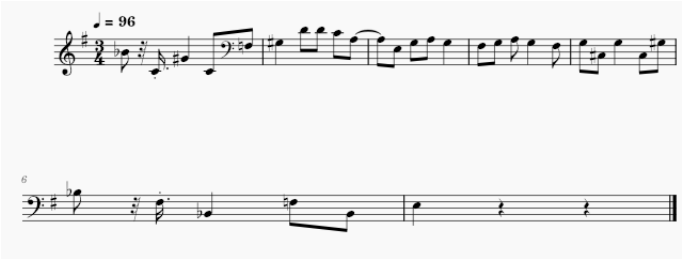


Fig. 65: Melody example created when the macrostructure [C, G, C] is set

macrostructure is not reliable enough, in the same dataset melodies ending in the G tonal sphere (dominant) are created, because of the wrapping process in the PGE algorithm.

Future work will involve evaluating the macrostructure of melodies by replacing the probability value updated for each rule iteration with a probability array. In this array, each rule will have a different probability based on its current depth and position within the melody's structure.

As demonstrated, the genetic algorithm generates melodies where patterns from the dataset remain recognizable, aligning with the primary characteristics of genetic algorithms. The fast convergence and high sensitivity of fitness functions make the genetic algorithm an excellent instrument for testing purposes, as evidenced by Fig. 58, where a fitness function evaluating interval presence in the

melody shows rapid convergence. Similarly, in Fig. 57, the fitness function strongly rewards melodies with smaller intervals and narrower pitch ranges.

Significantly different results were obtained when testing the LSTM network. This model was trained solely on note pitches, with rhythm added to the melody post-prediction using typical mazurka rhythmic figures (as depicted in Fig. 62). The resulting melodies exhibit key pitch characteristics of Chopin's mazurkas, including the frequent use of intervals of thirds, fifths, and chromaticisms, with pitch ranges typically falling between the C4 and A5. These results can be further evaluated using the "Analyst" tool presented earlier (Fig. 66). As depicted in Fig. 66, the neural network consistently favors intervals of a second and exhibits a pitch range predominantly above C5 (MIDI number 72).

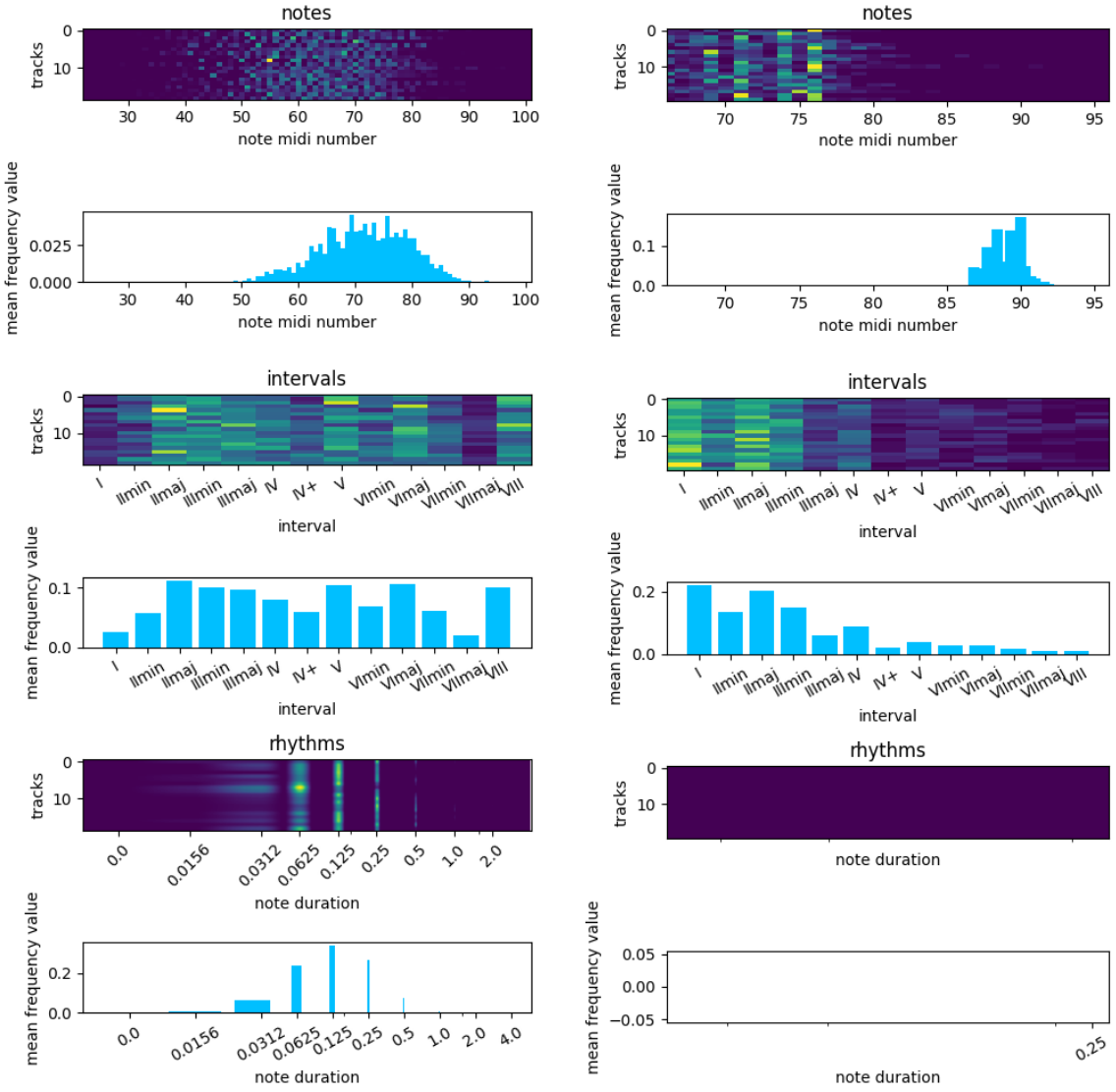


Fig. 66: Extracted musical characteristics from Chopin's mazurkas (a) and generated melodies by LSTM (b)

9 CONCLUSION

This work aims to serve as a testing "playground" for new methods in automatic music composition and analysis, leveraging the vast potential of MIDI files. The first part provides an overview of the historical advancements in automatic music composition up to recent years. It outlines the main characteristics, goals, and limitations of contemporary automatic composition algorithms. The focus is on two primary approaches to music composition: evolutionary algorithms and neural networks.

From a musical perspective, an objective of this study was to explore how the search for optimal composition algorithms could benefit the contemporary musical landscape. Through an examination of historically significant music analysis methodologies, it was observed that certain musical characteristics can indeed be quantified and computed. The results of this analysis, showcased through graphs illustrating distribution values and mean densities, provide valuable insights and are pertinent from a musicological standpoint.

Inspired primarily by Schenker's and Janáček's theoretical works concerning the decomposition of musical pieces to uncover their fundamental structures, the second tool developed in this work attempts to compose melodies based on core characteristics extracted from a melody dataset or according to the author's requirements. Earlier investigations in automatic music composition have pinpointed a significant obstacle: the creation of music possessing a structured format (such as the tonic-dominant-tonic structure prevalent in Classical music of the Classic period) and where its elements are clearly interlinked. In this work a way to define the macrostructure of a melody and create melodies according to it was presented.

The implementation of probabilistic grammar evolution presented here is ready for future enhancements. By making the probability values of grammar rules a multidimensional array, each rule could have varying probabilities of occurring based on its position in time. Another fundamental upgrade involves developing better fitness functions, composing melodies with varied pauses, and expanding the range of musical elements (different instruments, multiple voices, various rhythms, etc.) used in compositions.

Beyond application upgrades and developments, future work should focus on the numerous ways technological advancements can serve as tools for creating new art by composers, as analysis tools for musicologists, or as aids for music students and artists. MIDI files offer immense opportunities within the realm of tonality and equal temperament, examples of which are presented in this work. However, this starting point limits the research from engaging with electronic, experimental, and contemporary music, genres that are currently flourishing. Therefore, the algorithms

and tools presented should also be tested and developed for raw audio files to create new sounds, inspire new musical ideas, and better understand experimental and electronic music composed since the 1950s.

BIBLIOGRAPHY

[1] Akbari, M. and Liang, J. Semi-Recurrent CNN-based VAE-GAN for Sequential Data Generation. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. Institute of Electrical and Electronics Engineers Inc. june 2018, 2018-April, p. 2321–2325. DOI: 10.1109/ICASSP.2018.8461724. ISSN 15206149. Available at: https://arxiv.org/abs/1806.00509v1.

[2] Bachcentral. *A Johann Sebastian Bach Midi Page*. 2018. [cited 2024-05-22]. Available at: https://www.bachcentral.com/midiindexcomplete.html.

[3] Back, T., Hammel, U. and Schwefel, H. P. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*. 1997, vol. 1, p. 3–17. DOI: 10.1109/4235.585888. ISSN 1089778X.

[4] Bagavathi, C. and Saraniya, O. Evolutionary Mapping Techniques for Systolic Computing System. *Deep Learning and Parallel Computing Environment for Bioengineering Systems*. Academic Press. january 2019, p. 207–223. DOI: 10.1016/B978-0-12-816718-2.00020-8.

[5] Biles, J. A. GenJam: An interactive genetic algorithm jazz improviser. *The Journal of the Acoustical Society of America*. 1997, vol. 102, 5_Supplement, p. 3181–3181. DOI: 10.1121/1.420841. ISSN 0001-4966.

[6] Bjørndalen. *Standart MIDI Files*. 2023. [cited 2024-05-22]. Available at: https://mido.readthedocs.io/en/stable/files/midi.html.

[7] Both, C. *The influence of concepts of information theory on the birth of electronic music composition : Lejaren A. Hiller and Karlheinz Stockhausen, 1953-1960*. 1995. ISBN 0612083047.

[8] Bresin, R. Artificial neural networks based models for automatic performance of musical scores. *Journal of New Music Research*. Taylor & Francis. 1998, vol. 27, no. 3, p. 239–270.

[9] Chopin, F. Mazurkas op. 17. In: Chopin, I. F., ed. *Dzieła wszystkie Fryderyka Chopina*. Warsawa: Polskie Wydawnictwo Muzyczne, 1949 (1834), X: Mazurkas, p. 26–35. Available at: https://imslp.org/wiki/Mazurkas,_Op.17_(Chopin,_Fr%C3%A9d%C3%A9ric).

[10] Chopin, F. Polonaise-Fantasie op. 61. In: Chopin, I. F., ed. *Dzieła wszystkie Fryderyka Chopina*. Warsawa: Polskie Wydawnictwo Muzyczne, 1949 (1846),

VIII: Polonaises, p. 70–87. Available at: https://imslp.org/wiki/Polonaise-fantaisie,_Op.61_(Chopin,_Fr%C3%A9d%C3%A9ric).

[11] Chopin, F. Sonata n. 2, op. 35. In: Chopin, I. F., ed. *Dzieła wszystkie Fryderyka Chopina*. Warsawa: Polskie Wydawnictwo Muzyczne, 1950 (1839), VI: Sonatas, p. 46–70. Available at: https://imslp.org/wiki/Piano_Sonata_No.2,_Op.35_(Chopin,_Fr%C3%A9d%C3%A9ric).

[12] Coenen, A. David Cope, Experiments in Musical Intelligence. A-R Editions, Madison, Wisconsin, USA. Vol. 12 1996. *Organised Sound*. 1997, vol. 2, no. 1, p. 57–60. DOI: 10.1017/S1355771897210101.

[13] Cook, N. *Analysing Musical Multimedia*. Oxford: Oxford University Press, 1998. 290 p. ISBN 0-19-816737-7.

[14] Despois, J. *Finding the genre of a song with Deep Learning-AI Odyssey part. 1*. December, 2018. [cited 2024-05-22]. Available at: https://hackernoon.com/finding-the-genre-of-a-song-with-deep-learning-da8f59a61194.

[15] Dong, H.-W., Hsiao, W.-Y., Yang, L.-C. and Yang, Y.-H. MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. In: *AAAI Conference on Artificial Intelligence*. 2017. DOI: 10.48550/arXiv.1709.06298.

[16] Eigenfeldt, A., Thorogood, M., Bizzocchi, J. and Pasquier, P. MediaScape: towards a video, music, and sound metacreation. *Journal of Science and Technology of the Arts*. Universidade Católica Portuguesa. 2014, vol. 6, no. 1. DOI: 10.7559/citarj.v6i1.129. ISSN 1646-9798.

[17] Fredericks, E. M., Diller, A. C. and Moore, J. M. Generative Art via Grammatical Evolution. In:. Institute of Electrical and Electronics Engineers Inc., 2023, p. 1–8. DOI: 10.1109/GI59320.2023.00010. ISBN 9798350312324.

[18] Freyberg, K. *Introducing v3*. 2024. [cited 2024-05-22]. Available at: https://suno.com/blog/v3.

[19] Friberg, A. Generative rules for music performance: A formal description of a rule system. *Computer Music Journal*. JSTOR. 1991, vol. 15, no. 2, p. 56–71.

[20] Gang, D. and Berger, J. Modeling the degree of realized expectation in functional tonal music: A study of perceptual and cognitive modeling using neural networks. In: Citeseer. *Proceedings of the International Computer Music Conference*. 1996, p. 454–457.

[21] Garton, B., Todd, B. R. P. and Loy, D. G. Artificial Intelligence Music and Connectionism. *Artificial Intelligence*. 1995, vol. 79, p. 387–398.

[22] GoogleAI. *Magenta*. 2024. [cited 2024-05-22]. Available at: `https://magenta.tensorflow.org/research`.

[23] Granroth Wilding, M. and Steedman, M. A Robust Parser-Interpreter for Jazz Chord Sequences. *Journal of new music research*. Lisse: Routledge. 2014, vol. 43, no. 4, p. 355–374. DOI: 10.1080/09298215.2014.910532. ISSN 0929-8215.

[24] Hadjeres, G., Pachet, F. and Nielsen, F. DeepBach: a Steerable Model for Bach Chorales Generation. *34th International Conference on Machine Learning, ICML 2017*. International Machine Learning Society (IMLS). december 2016, vol. 3, p. 2187–2196. DOI: 10.48550/arXiv.1612.01010. Available at: `https://arxiv.org/abs/1612.01010v2`.

[25] Hiller, L. and Isaacson, L. *Experimental Music: Composition with an Electronic Computer*. Greenwood Press, 1979. 197 p. ISBN 9780313221583.

[26] Huang, Y. S. and Yang, Y. H. Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions. *MM 2020 - Proceedings of the 28th ACM International Conference on Multimedia*. Association for Computing Machinery, Inc. february 2020, p. 1180–1188. DOI: 10.1145/3394171.3413671. Available at: `https://arxiv.org/abs/2002.00212v3`.

[27] Husbands, P., Copley, P., Eldridge, A. and Mandelis, J. An introduction to evolutionary computing for musicians. In: Miranda, E. R., ed. *Evolutionary computer music*. London: Springer, 2007, p. 1–27. DOI: 10.1007/978-1-84628-600-1_1. ISBN 978-1-84628-600-1.

[28] Isaacson, E. Neural network models for the study of post-tonal music. In: Leman, M., ed. *Music, Gestalt, and Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, p. 237–250. ISBN 978-3-540-69591-2.

[29] Jacob, B. Composing With Genetic Algorithms. *Proceedings of the 1995 International Computer Music Conference*. San Francisco: ICMA. june 1995.

[30] Janáček, L. *The Published Theoretical Works of Leoš Janáček*. Brno: Editio Janáček, 2020. 685 p. ISBN 978-80-904052-7-1.

[31] Jeong, J., Kim, Y. and Ahn, C. W. A multi-objective evolutionary approach to automatic melody generation. *Expert systems with applications*. New York:

Elsevier Ltd. 2017, vol. 90, p. 50–61. DOI: 10.1016/j.eswa.2017.08.014. ISSN 0957-4174.

[32] Jin, C., Tie, Y., Bai, Y., Lv, X. and Liu, S. A Style-Specific Music Composition Neural Network. *Neural Processing Letters*. Springer. december 2020, vol. 52, p. 1893–1912. DOI: 10.1007/s11063-020-10241-8. ISSN 1573773X.

[33] Jędrzejewska, M. K., Zjawiński, A. and Stasiak, B. Generating Musical Expression of MIDI Music with LSTM Neural Network. In: *2018 11th International Conference on Human System Interaction (HSI)*. 2018, p. 132–138. DOI: 10.1109/HSI.2018.8431033.

[34] Katoch, S., Chauhan, S. S. and Kumar, V. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*. Springer. february 2021, vol. 80, p. 8091–8126. DOI: 10.1007/s11042-020-10139-6. ISSN 15737721.

[35] Keras. *Keras 3 API documentation*. 2024. [cited 2024-05-22]. Available at: `https://keras.io/api/`.

[36] Kochenderfer, M. J. and Wheeler, T. A. *Algorithms for Optimization*. The MIT Press, 2019. ISBN 0262039427.

[37] Lattner, S. and Kepler, J. Modeling Musical Structure with Artificial Neural Networks. january 2020. DOI: 10.48550/arXiv.2001.01720. Available at: `https://arxiv.org/abs/2001.01720v1`.

[38] Leung, K. *Text-to-Audio Generation with Bark, Clearly Explained*. 2023. [cited 2024-05-22]. Available at: `https://betterprogramming.pub/text-to-audio-generation-with-bark-clearly-explained-4ee300a3713a`.

[39] Lim, Y.-Q., Chan, C. S. and Loo, F. Y. Style-Conditioned Music Generation. In: *2020 IEEE International Conference on Multimedia and Expo (ICME)*. 2020, p. 1–6. DOI: 10.1109/ICME46284.2020.9102870.

[40] Liu, C. H. and Ting, C. K. Computational Intelligence in Music Composition: A Survey. *IEEE Transactions on Emerging Topics in Computational Intelligence*. Institute of Electrical and Electronics Engineers Inc. february 2017, vol. 1, p. 2–15. DOI: 10.1109/TETCI.2016.2642200. ISSN 2471285X.

[41] Manzelli, R., Thakkar, V., Siahkamari, A. and Kulis, B. Conditioning Deep Generative Raw Audio Models for Structured Automatic Music. *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018*. International Society for Music Information Retrieval. june 2018, p. 182–189. Available at: `https://arxiv.org/abs/1806.09905v1`.

[42] Marquetti, A. *Solos (dice game) and conductor (neural network)*. ProQuest Dissertations Publishing, 2015.

[43] Mastropasqua, M. PerchÃĬ la musica di Webern ÃÍ cosÃň interessante da analizzare? In: Casadei, M. T. M., ed. *Anton Webern. Un punto, un cosmo*. Lucca: Libreria Musicale Italiana, 1998. ISBN 9788870962024.

[44] Mégane, J., Lourenço, N. and Machado, P. Probabilistic Structured Grammatical Evolution. may 2022. DOI: 10.1109/CEC55065.2022.9870397. Available at: http://arxiv.org/abs/2205.10685http://dx.doi.org/10.1109/CEC55065.2022.9870397.

[45] Midiworld. *Midiworld.com*. 2024. [cited 2024-05-22]. Available at: https://www.midiworld.com/chopin.htm.

[46] Miranda, E. R. and Biles, J. A. *Evolutionary Computer Music*. 1. Aufl.th ed. London: Springer Verlag London Limited, 2007. ISBN 1846285992.

[47] Miranda, E. R., Correa, J. and Wright, J. Categorising complex dynamic sounds. *Org. Sound*. USA: Cambridge University Press. aug 2000, vol. 5, no. 2, p. 95âĂŞ102. DOI: 10.1017/S1355771800002065. ISSN 1355-7718. Available at: https://doi.org/10.1017/S1355771800002065.

[48] OpenAI. *MuseNet*. 2019. [cited 2024-05-22]. Available at: https://openai.com/research/musenet.

[49] Ortega, A., Cruz, M. de la and Alfonseca, M. Christiansen Grammar Evolution: Grammatical Evolution With Semantics. *IEEE transactions on evolutionary computation*. New York, NY: IEEE. 2007, vol. 11, no. 1, p. 77–90. DOI: 10.1109/TEVC.2006.880327. ISSN 1089-778X.

[50] Pelchat, N. and Gelowitz, C. Neural Network Music Genre Classification. *Canadian Journal of Electrical and Computer Engineering*. july 2020, vol. 43, p. 170–173. DOI: 10.1109/cjece.2020.2970144.

[51] Pelikan, M., Goldberg, D. E. and Lobo, F. G. A Survey of Optimization by Building and Using Probabilistic Models. *Computational optimization and applications*. New York: Springer Nature B.V. 2002, vol. 21, no. 1, p. 5–20. DOI: 10.1023/a:1013500812258. ISSN 0926-6003.

[52] Pettorossi, A. *Automata Theory and Formal Languages*. Springer Cham, 2022. 280 p. ISBN 978-3-031-11964-4.

[53] Prisco, R. D., Zaccagnino, G. and Zaccagnino, R. Evocomposer: An evolutionary algorithm for 4-voice music compositions. *Evolutionary Computation*. MIT Press Journals. 2019, vol. 28, p. 489–530. DOI: 10.1162/evco_a_00265. ISSN 15309304.

[54] Prisco, R. D. and Zaccagnino, R. Creative DNA computing: splicing systems for music composition. *Soft Computing*. Springer Science and Business Media Deutschland GmbH. september 2022, vol. 26, p. 9689–9706. DOI: 10.1007/s00500-022-06828-z. ISSN 14337479.

[55] Riemann, H. Form-giving Principles: Harmony and Rhythm. In:. Cambridge University Press, February 2014, p. 23–43. DOI: 10.1017/cbo9781139542531.003.

[56] Roberts, A., Engel, J., Raffel, C., Hawthorne, C. and Eck, D. A hierarchical latent vector model for learning long-term structure in music. In: PMLR. *International conference on machine learning*. 2018, p. 4364–4373.

[57] Rosen, C. *The Romantic Generation*. Milano: Adelphi, 1997. 791 p. ISBN 8845913392.

[58] Ryan, C., OÂĂŹNeill, M. and Collins, J. J. *Handbook of grammatical evolution*. Springer International Publishing, january 2018. 1-497 p. ISBN 9783319787176.

[59] Sarmento, P., Kumar, A., Chen, Y.-H., Carr, C., Zukowski, Z. et al. GTR-CTRL: Instrument and Genre Conditioning for Guitar-Focused Music Generation with Transformers. In: Johnson, C., Rodríguez Fernández, N. and Rebelo, S. M., ed. *Artificial Intelligence in Music, Sound, Art and Design*. Cham: Springer Nature Switzerland, 2023, p. 260–275. DOI: 10.1007/978-3-031-29956-8_17. ISBN 978-3-031-29956-8.

[60] Saxena, D. and Cao, J. Generative Adversarial Networks (GANs). *ACM Computing Surveys (CSUR)*. ACMPUB27New York, NY, USA. may 2021, vol. 54. DOI: 10.1145/3446374. ISSN 15577341. Available at: `https://dl.acm.org/doi/10.1145/3446374`.

[61] Schenker, H. and Salzer, F. *Five Graphic Music Analyses (Fünf Urlinie-Tafeln)*. Dover Publications, 1969. Dover books on music. ISBN 9780486222943. Available at: `https://books.google.cz/books?id=GrgwubvmFswC`.

[62] Schlüter, J. and Böck, S. Improved musical onset detection with convolutional neural networks. In: IEEE. *2014 ieee international conference on acoustics, speech and signal processing (icassp)*. 2014, p. 6979–6983.

[63] Scirea, M., Togelius, J., Eklund, P. and Risi, S. Affective evolutionary music composition with MetaCompose. *Genetic Programming and Evolvable Machines*. Springer New York LLC. december 2017, vol. 18, p. 433–465. DOI: 10.1007/s10710-017-9307-y. ISSN 1389-2576. Available at: http://link.springer.com/10.1007/s10710-017-9307-y.

[64] Simpson, A. J., Roma, G. and Plumbley, M. D. Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network. In: Springer. *Latent Variable Analysis and Signal Separation: 12th International Conference, LVA/ICA 2015, Liberec, Czech Republic, August 25-28, 2015, Proceedings 12*. 2015, p. 429–436.

[65] Skúli, S. *How to Generate Music using a LSTM Neural Network in Keras*. 2017. [cited 2024-05-22]. Available at: https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5.

[66] Skúli, S. *Classical-Piano-Composer*. 2019. [cited 2024-05-22]. Available at: https://github.com/Skuldur/Classical-Piano-Composer.

[67] Tatar, K., Bisig, D. and Pasquier, P. Latent Timbre Synthesis: Audio-based variational auto-encoders for music composition and sound design applications. *Neural Computing and Applications*. Springer Science and Business Media Deutschland GmbH. january 2021, vol. 33, p. 67–84. DOI: 10.1007/s00521-020-05424-2. ISSN 14333058.

[68] Vechtomova, O. and Sahu, G. LyricJam Sonic: A Generative System for Real-Time Composition and Musical Improvisation. In: *Artificial Intelligence in Music, Sound, Art and Design: 12th International Conference, EvoMUSART 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12âĂŞ14, 2023, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2023, p. 292âĂŞ307. DOI: 10.1007/978-3-031-29956-8_19. ISBN 978-3-031-29955-1. Available at: https://doi.org/10.1007/978-3-031-29956-8_19.

[69] Wang, Y. Music Composition and Emotion Recognition Using Big Data Technology and Neural Network Algorithm. *Computational Intelligence and Neuroscience*. Hindawi Limited. 2021, vol. 2021. DOI: 10.1155/2021/5398922. ISSN 16875273.

[70] Waskan, J. *Connectionism*. 2022. 22/05/2024. Available at: https://iep.utm.edu/connectionism-cognition/.

[71] Wiafe, A., Nutrokpor, C., Owusu, E., Kastriku, F. A. and Wiafe, I. Using genetic algorithms for music composition: implications of early termination on aesthetic quality. *International Journal of Information Technology*

(Singapore). Springer Science and Business Media B.V. june 2022, vol. 14, p. 1875–1881. DOI: 10.1007/s41870-022-00897-x. ISSN 25112112.

[72] Zhong, Y. and Xiang, Y. Design and Realization of Music Recognition based on Speech Recognition. 2011. DOI: 10.1109/NCIS.2011.110.

SYMBOLS AND ABBREVIATIONS

AMC Automatic music composition

NN Neural Network

EA Evolutionary algorithm

CNN Convolutional neural network

LTS Long term SÃňstructures

LSTM Long short term memory neural network

VAE Variational Auto-Encoder

RNN Recurrent neural network

GPT Generative pre-trained transformer

GAN Generative adversial networks

AC Autor-critic network(Hadjeres, 2016)

RBM Restricted Boltzmann machine

GE Grammatical evolution

PGE Probabilistic grammatical evolution

CFG Context free grammar

S Start symbol (axiom)

T Terminal symbol

NT Non-Terminal symbol

BNF Backus-Naur Form (grammars)

LIST OF FIGURES

LIST OF TABLES