Departamento de Matemática Aplicada a las Tecnologías de la Información y las Telecomunicaciones

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Universidad Politécnica de Madrid

Generation of Jazz Improvisations in MATLAB

Álvaro Sánchez Hidalgo

Final Project: Grado en Ingeniería de Software (Plan 2009)

Year 2016-17

Director: Fráncisco Gómez Martín

Contents

Chapter 1

Introduction

1.1 Goals and motivations

It is often said that "a genius is made, not born." However, creative processes are often considered as acts that require some kind of innate gift, sometimes mystical in its nature. Not surprisingly, music is one of those activities, where even practitioners themselves are unsure of where their compositions come from, and this is despite their spending years working to become masters of their instruments.

One can affirm that our ability to quickly adapt our past experiences for new contexts is one of the traits that defines human intelligence. Unfamiliar situations arise where we have to draw upon our judgment, applying the course of action that we deem most appropriate basing our decision on previous occurrences and taking into account new factors that make the circumstances different in order to respond in a fitting way. Such a problem also emerges in music, particularly in the jazz genre, where improvisation is a core feature.

Improvisation emerges from combining creativity and adaptation. It may well be defined as an immediate musical composition. It is one of the most difficult abilities to master as it demands a complete command of the instrument as well as exceptional creativity and musical thought. It forces the artist to adapt her playing at an extraordinarily fast pace, while they move within the confines of the song that are being built, in real time, by the rest of the band. In jazz, this complexity is aggravated by the unconventional chord progressions and frequent key changes (unconventional understood as deviation from the common practice).

No matter how challenging and intricate a solo happens to be, improvisation is a skill that can be learned. Proficient jazz artists such as John Coltrane or Dizzy Gillespie would not have felt the need to practise scales everyday for decades if that were not the case. Nevertheless, it is unclear how this skill is actually acquired. Some experts contend that solos are the result of implicit musical rules that dictate the sequence of notes that shapes

the output. Others, however, argue that the continued repetition of similar material leads to the appearance of patterns in the soloist's work, sometimes spanning years of their career.

This document will touch upon previous research that explores both options, as well as go through some examples of commercial and academical software that tackle the topic of improvisation from different standpoints.

We will focus on the former theory and will go into detail about the implementation of a pattern-based algorithm for the generation of solos based on the works of jazz saxophonist Charlie Parker, whose transcribed works are easily available on the internet and other sources. This algorithm, originally presented in Norgaard et al. (2013), will employ Markov chains, a concept that will be thoroughly described throughout the document, to generate a sequence of notes that bear a resemblance to Parker's material, partially through the appearance of reoccurring musical motifs. In addition, our improved program will offer the possibility of inputting chord progressions to constrain the output of the program through a user interface.

We hope this work will help shed some light on the ongoing debate between the aforementioned schools of thought. Improvisation is not just a fascinating musical problem, but also a part of our everyday lives. A better understanding of improvisation is a better understanding of the human mind, and every step towards comprehending our own thinking helps us find the answer one of the most important questions for mankind: how does our brain work?

1.2 Objetivos y motivación

A menudo se dice que "el genio no nace, se hace". Sin embargo, a menudo se considera que los procesos creativos requieren alguna forma de don innato, a veces místico en su naturaleza. La música es una de estas actividades, donde incluso aquellos que la practican no están seguros de donde vienen sus composiciones a pesar de haber empleado años de trabajo en convertirse en maestros de sus instrumentos.

Uno puede afirmar que nuestra habilidad para adaptar rápidamente nuestras experiencias pasadas a nuevos contextos es una de las características que define la inteligencia humana. En nuestra vida aparecen situaciones que no nos son familiares donde tenemos que recurrir al juicio, aplicando el curso de acción que consideremos más apropiado y basando nuestra decisión en sucesos previos y teniendo en cuenta los nuevos factores que hacen que las circunstancias sean diferentes con el fin de responder de manera correcta. Tal problema emerge también en la música, en particular en el género del jazz, donde la improvisación es un rasgo clave.

No importa lo complicado que resulte ser un solo, la improvisación es una habilidad que puede ser aprendida. Sin embargo, no está claro cómo se adquiere en realidad esta habilidad. Una de las teorías imperantes sugiere que la repetición continuada de material

similar lleva a la aparición de patrones en la obra del solista, que a menudo abarca años de su carrera.

Nos basaremos en esta teoría y la exploraremos en detalle para a continuación describir y ampliar la implementación de un algoritmo para la generación de solos a partir del corpus del saxofonista Charlie Parker. El algoritmo empleará cadenas de Markov para producir una secuencia de notas que guardarán una semejanza al material de Parker debido a la aparición de motivos musicales recurrentes. Además se ofrecerá al usuario la posibilidad de insertar sus propias progresiones de acordes para restringir la salida del programa a través de una interfaz de usuario.

Esperamos que esto aporte algo de luz al debate de la improvisación. La improvisación no es solamente un problema musical fascinante, sino también una parte de nuestra vida cotidiana. Un mejor entendimiento de la improvisación conlleva un mejor entendimiento de la mente humana, y todo paso hacia la comprensión de nuestro propio pensamiento nos ayuda a encontrar la respuesta a una de las preguntas más importantes de la humanidad: ¿cómo funciona nuestro cerebro?

1.3 Structure of this document

The present document is divided up into five parts:

- Following this introduction, Section 2 will explore the problem of improvisation and provide an overview on the two contrasting explanations for the essence of improvisation, with a rundown of the existing literature on the matter. The main concepts that will be used in later sections will also be presented.
- Section 3 will examine some of the algorithms already available for the generation of improvisations as well as the state of the art in software products that produce improvised output in compliance to one of the theories already illustrated.
- Section 4 will describe our algorithm in detail. First, its original state will be analyzed in order to expose its limitations; subsequently, the improvements that have been added in order to at least partially remove these will be thoroughly presented.
- In Section 5, the output of the algorithm will be evaluated in order to study its similarity with Charlie Parker's corpus, as well as compared to other existing algorithms.
- Finally, in Section 6, the conclusions of this project will be presented, along with some hints about where future research in this subject may be headed.

Chapter 2

Problem

2.1 The nature of improvisation

The Oxford Dictionary of Music (Rutherford-Johnson et al., 2012) defines improvisation as a "performance according to the inventive whim of the moment.'. Such a definition could be deemed as trivial or frivolous, but there is a great amount of meaning hidden behind those words.

The basis of improvisation is dialogue: with musicians, with the audience, with the music itself. Jazz works often take the form of a conversation between instruments. In a sense, the point is that improvisation is a collaborative act where the environment plays an integral role. Soloists take turns to contribute to the creation of a coherent piece, and the many variables associated with it (from the particular mood of the performers to the acoustics of the room) imply that it is impossible to recreate the exact same solo twice.

As a result, jazz music has improvisation at its core. Jazz pieces are often lengthy creations that are improvised to a large extent, with the performers only aware of the general structure of the piece (aspects like the key the main phrases or the form, which are quite predetermined). The changes in direction that come along the way and that are consequence of the many variables involved in the musical process are impossible to predict; therefore, the musicians have to be constantly alert to them. There are usually several solos in the same piece, each of them exhibiting distinct features of the person playing it, and each of them being created in real time to suit the feel of the work, without relying on a printed score.

To improvise, then, means to simultaneously compose and execute music. Being able to seamlessly incorporate a solo to an ongoing musical performance requires, obviously, a remarkable skill. It is unknown how exactly the brains of these virtuosos acquire that skill. However, two theories have been postulated by music theorists that attempt to explain how an improvised output comes to be. We will look at these theories in detail in the next two sections.

2.1.1 Motor patterns

The mastery of an instrument is hardly an innate skill; quite the contrary, it is the result of years and years of practice. Even jazz prodigies like John Coltrane or Miles Davis spent upwards of ten hours a day rehearsing to achieve their exceptional dexterity. It can be stated then that any performer who is proficient enough with an instrument that improvisation comes naturally to them must have invested large amounts of effort and time practising, often by means of repetition in a goal-oriented practice.

Repetition is a key factor in the learning process. Soloists that have complete control over their instrument are not just able to remember sequences of notes in a particular melody, but also the movements that are necessary to play it. These movements are known as **motor patterns**, and they are the result of muscle memory. This memory is often accessed unconsciously, just like a person who is told to sign his name does not need to stop and think about what they are doing. In fact, if we try to deliberately replicate it we often fail to do so, or at least it is done more slowly.

This poses a problem: how is that particular pattern adapted in order to fit in different pieces? Schmidt (1975) developed the concept of **schema**. A schema is defined as a set of stimuli that has to be adapted in order to fit a new situation. These stimuli may be based on the current circumstances (in our example, that could be the key of the song or the chord that is playing) as well as past experiences (such as solos that have already been performed by the player). Essentially, one remembers old movements and their outcomes, and modifies those specifications to produce new outcomes that can be used in the future to repeat this process.

This holds true for improvisation as well. If we observe the improvised solos of any sufficiently talented and prolific musician, we will encounter a number of patterns that are repeated, with some minor adjustments, across the whole corpus. These can be melodic, such as a short sequence of pitch intervals (number of semitones between two pitches), or rhythmic, like the relative duration of a series of notes. These patterns have been internalized by the performer through repetition and eventually become distinctive as a result of their recurrence in their body of work.

From a more formal point of view, we can model this situation with Markov chains. A Markov chain is used to simulate a process where we have a finite set of states and transition probabilities between them. In our specific case, a state in a Markov chain may represent the last k notes that have been played. This is the same as saying that we are employing a Markov chain of order k. How Markov chains are used to generate a solo in the style of a given corpus will be treated in following sections. What is relevant is that the presence of repeated patterns indicates a higher probability of these patterns being reproduced by proceeding along the Markov chain, since the transition probabilities can be obtained by an analysis of the corpus. This generation is the result of a random walk: we start on an arbitrary state and move to another according to those probabilities.

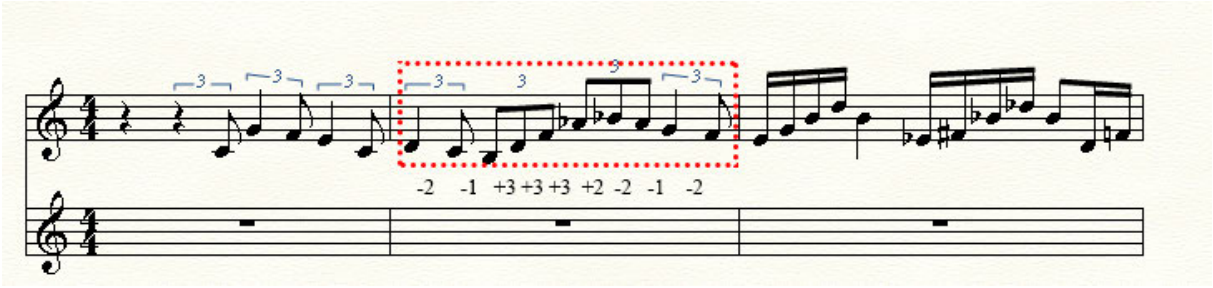


Figure 2.1: Pattern of 9 pitch intervals appearing in the second measure of the Parker corpus.

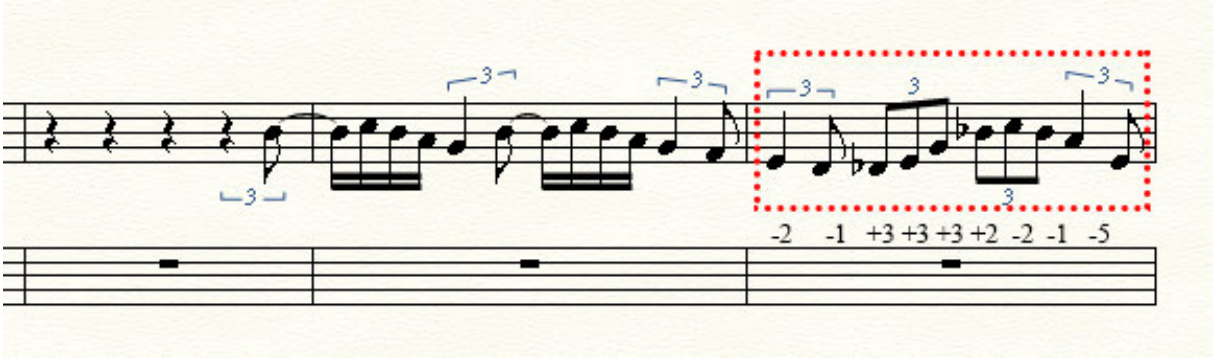


Figure 2.2: An almost identical pattern to the one in 2.1, that appears in a different solo.

2.1.2 Rule-based procedures

A possible argument against the theory above is that is fails to explain where did those patterns come from in the first place. Even if they are the result of imitating other soloists, the motifs that appear all over some jazz musicians' work must have been invented from square one at some point. This point of contention is exposed by Johnson-Laird (2002).

That article argues for an algorithmic generation of melodies that is not dependent upon working memory. According to the motor patterns theory, working memory is responsible for storing those patterns during the improvisation so they can be accessed at any given time and adapted to fit the current constraints. This manipulation is key to understanding the difference between working memory and short-term memory, since the latter does not allow for the transformation of the stored material. The problem with storing patterns in working memory is that access to it is too slow, so it would be impossible for a musician to adapt these patterns in real time.

However, reuse is still supported by this theory, but it is long-term memory that is involved in storing the motifs. At any rate, the driving force behind improvisation according to this theory is the use of scales that fit the underlying chord. Musicians who

have trained for years have developed an intuition for what scale is suggested by a given chord progression or a musical mode. This, together with concepts such as pitch contour, syncopation or harmony, allows generation of melodies by human players. As a result, a program with that same knowledge could then apply it to algorithmically generate an infinite number of melodies. How a program learns which scales fit which chords, and especially which notes within the scale are the most appropriate at any given time, is a delicate matter of discussion.

Temperley (2007) proposes a solution to that problem. The analysis of a corpus can yield important results, such as the distribution of pitches, distances between them, and degrees within a key. The combination of those results can produce a probabilistic model that can be used for solo generation that accounts for note proximity as well as harmonic constraints.

2.1.3 How does improvisation actually work?

Both approaches have their benefits and their drawbacks. A motor patterns-based solution is algorithmically easy to implement and generates solos that should be similar to the player whose corpus is being analyzed, but applying tweaks such as following chord progressions does not come naturally. On the other hand, a rule-based method is loose enough that making changes can be as simple as combining new probability distributions that participate in the generation of the output, but as a result the previous analysis of the corpus to obtain such probabilities becomes much more complex, and the results may not be as easily identified with a given soloist.

The solution could be a mixture between the two methods. The manner in which these techniques can be fused has not been explored much in the literature. Gómez et al. (2016) studies the concept of melodic similarity in flamenco to categorize musical pieces belonging to different sub-styles within the genre. This is achieved through the combination of observed rules as well as patterns obtained via analysis. This is substantially different from the problem of jazz improvisation, although it opens up the possibility of a merge between the two approaches, and that could be the direction where future research is headed.

Chapter 3

State of the art

In this chapter, a few algorithms and software products for generation of melodies will be surveyed.

3.1 Rule-based algorithm: Temperley

Temperley (2007) presents probabilistic models that attempt to solve the problems of rhythm and pitch generation within melodies. We will explore both of these in more detail.

3.1.1 Rhythm model

Temperley considers three different levels of beats, which can be represented on a grid. Upper-level beats occur at the beginning of measures. The middle level is known as the **tactus** level: this is the most prominent one, which people can easily identify as the "beat" of the piece. Lower-lever beats are the fastest ones and they complete the metrical structure. The relationships between levels conform the time signature of the piece, as seen in Figure 3.1.

The metrical analysis of the corpus (how this is performed is also explained by Temperley) yields a number of important results that include the probabilities of the occurrence of different time signatures, as well as the chance of a note taking place on a particular subdivision of the beat. What matters here is that every value that serves as an input to the algorithm can be obtained through an exploration of the corpus.

To generate a metrical structure, first a grid is created and the time signature is chosen. Once the relationships between levels have been established, the middle (tactus) level is generated: beats are placed one after another. The distance between two consecutive beats is dictated by a distribution which mean is the distance between the two previous ones, starting at 700 milliseconds. As a result, they vary only slightly over the piece.

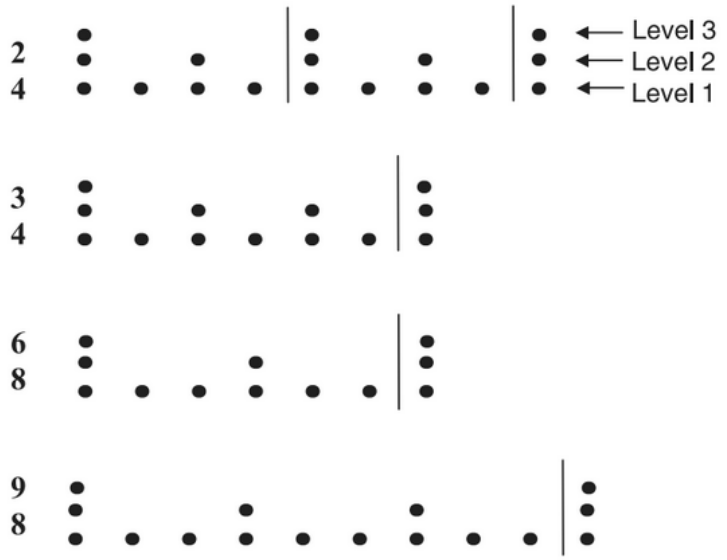


Figure 3.1: Four important time signatures and their corresponding grids.

Upper-level beats can be incorporated to the grid without any need of further operations, since they occur at the beginning of measures and we know the time signature. However, the placing of lower-level beats requires a distribution that ensures they happen halfway between two consecutive tactus beats (duple meter) or every third of the way (triple meter).

This completes the grid. We can now generate a sequence of note-onsets. According to the model, notes can only take place at pips, which are the smallest metrical subdivision, occurring every 50 milliseconds. The probability of a note occurring at a pip depends on the beat level of the pip (the higher the level, the more likely it is that there is a note there, although notes can be present at pips where there is no beat).

This process only generates a random sequence of locations where notes exist. The next section explores how to assign a pitch to those locations in a fitting way, accounting for key, range and distance between pitches.

3.1.2 Pitch model

Temperley presents models for both the monophonic and polyphonic cases: we will focus on the former. Like the rhythm model described earlier, the generation of a sequence of pitches revolves around a number of probability distributions that can be obtained directly from a corpus and that are combined in order to provide the probability of a particular pitch given a context.

The first one that is used is a distribution of pitches across the corpus. This is as simple as counting how many times each distinct pitch appears, and will give us a notion of the range of the instrument the melodies are being played on. The purpose of this information is supplying a central pitch around which the rest of the pitches will be placed.

Next is a distribution of intervals, the distance between two consecutive notes in semitones. If the pitch descends, the distance will be negative; if it rises, it will be positive; if it is zero, the note is repeated. Intervals also play a crucial role in our algorithm; however, here Temperley uses them in an isolated manner, and they are independent from the rest of factors before they are integrated into the final profile for the generation of melodies.

Finally, an analysis is conducted to determine **key-profiles**. These represent the probability of each of the twelve notes appearing given a particular key. Instead of using the absolute pitches (which would necessitate 24 profiles, 12 major and 12 minor), the model employs relative pitches, meaning that we only need two profiles.

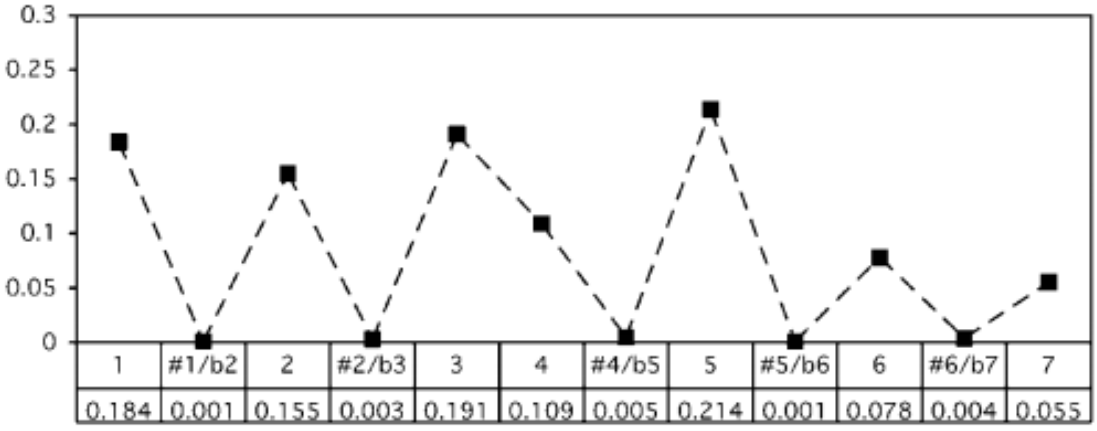


Figure 3.2: Key-profile for a major key. Note that the degrees of the major chord (1, 3 and 5) are the most frequent.

These three distributions can be combined into what Temperley calls an **RPK profile** (range-proximity-key). Given a central pitch, a previous pitch and a key, it is now possible to generate the probability distribution of the next pitch. This allows us to generate a sequence of pitches that meets our conditions.

Both algorithms can be considered examples of rule-based methods. The probabilities that follow from the analysis of the corpus are the result of the existence of tonal rules that the specific style (in Temperley's case, folk songs) generally adheres to. That does not mean patterns cannot take place, but they are entirely accidental. Since one note

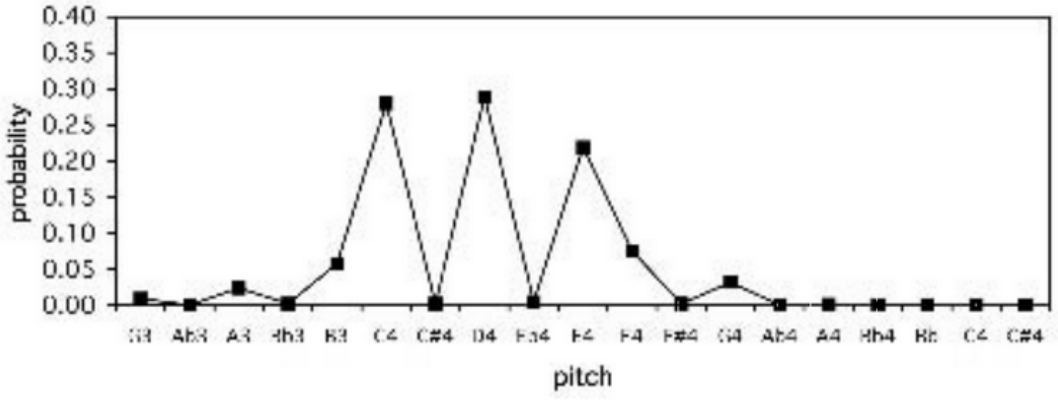


Figure 3.3: Probability of the next pitch for a central pitch of Ab4, a previous pitch of C4, and a key of C major.

only depends on the previous one, the appearance of motifs that were in the corpus to begin with is very unlikely.

These two models are relatively easy to build and they can be improved incorporating further variables (we could include the notion of **chord-profiles** in addition to the key-profiles, for instance) but they have their share of drawbacks. First, it is not clear that rhythm and pitch are independent: longer notes tend to belong to the underlying chord, for example. In addition, as a result of the purely rule-based approach, melodies generated with this method will not markedly resemble those from the original corpus. Consequently, despite it being algorithmically useful, its application to the study of improvisation is limited.

3.2 Pattern-based algorithm: Pachet

The sequential nature of patterns means that it is necessary to make use of a tool that allows the modeling of phenomena where the temporal dimension plays an important role. Markov chains are such a tool.

A Markov process is a memory-less stochastic process. In layman's terms, it is any process where the future states depends solely on the present. This means knowing the entire history of the process does not supply any more information, and as a result no memory is needed to store the past states.

Markov chains can be defined by a finite set of states and their corresponding transition probabilities. Transition probabilities may depend on one or more states: the number of states on which a future state depends on is known as the **order** of the Markov chain. Note that the fact that multiple states can dictate the next step does not mean that the

process is memory-dependant: that memory has a fixed size and a Markov chain of order $k > 1$ can be transformed into one of order 1 by the aggregation of said states.

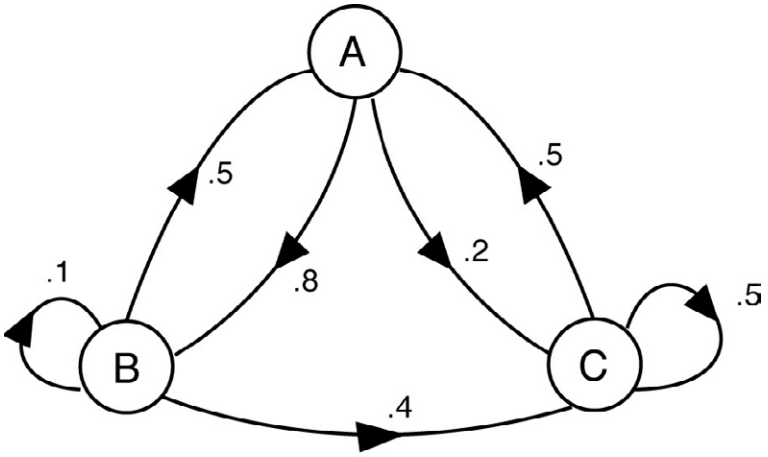


Figure 3.4: Markov chain with states A, B and C.

The application of Markov chains to the generation of musical output is almost direct. The analysis of a corpus yields the states (such as pitches or durations of notes) and the transitions between them. A starting state could be selected arbitrarily, and then the transition probabilities can be used to visit a random neighbour. The repetition of this procedure is called a **random walk**.

Pachet (2002) proposes a method to learn patterns from a musical input in real time, and then use a Markov approach to generate melodies that resemble that input. Instead of Markov chains, a prefix tree structure is used to index all the possible continuations to the sequence. The algorithm inspects the tree to check for a continuation to the longest possible subsequence; if there is none, it will continue the sequence with a note at random. This of course defeats the purpose of pattern-based generation, so further improvements have been introduced, such as the use of hidden Markov models.

Pachet & Roy (2011) expands the notion of Markov processes incorporating constraints. This allows some extent of user input in the random process. The phrases that result from the execution of the process will then fit some conditions set by the user, such as an underlying chord progression or a specific melodic contour. As a result, only sequences that match the restrictions will be generated. The cost of such an algorithm is expensive, so optimizing the production of sequences is critical.

These algorithms provide reasonably good results, even in real-time situations. However, they do not attack the rhythm problem naturally, and solutions that take into account the length of notes or placement of rests have to be incorporated separately

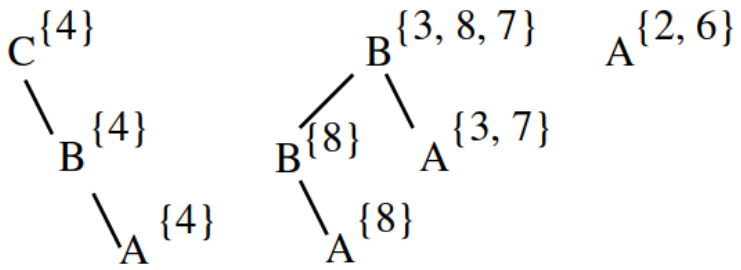


Figure 3.5: Prefix tree that results from inputting the sequences (A,B,C,D) and (A,B,B,C). The numbers between brackets are the continuation indexes, e.g., the continuation to "ABB" is the 8th note in the input, that is, C.

3.3 Available software

3.3.1 Impro-Visor

Impro-Visor (Keller, 2005) is a free software tool designed to help musicians learn to improvise. The user that is composing a solo receives feedback from the program, including suggestions to improve the melody as well as the user's ability to create new solos on the fly. The program is also capable of generating improvisations in the style of several jazz musicians, such as Bill Evans, John Coltrane or Keith Jarrett.

This is achieved through the use of probabilistic grammars. The program derives rules from the phrases within a corpus and assigns probabilities to them. This allows Impro-Visor to employ this set of rules in a generative manner. Rules can also include other rules, which supports the appearance of recurrence and transitions between rules, giving the grammar a structure similar to that of Markov chains.

Essentially, an instance of the application of those rules yields an **abstract melody**, a string of symbols that corresponds to the notes' categories (such as whether it is a tone within the chord, within the key, or neither) and durations. This abstract melody doesn't contain any actual notes: this means they can be used for any given chord progression just assigning the pitches that correspond to the particular categories, according to their probabilities.

The grammar for a specific corpus is the result of clustering these abstract melodies by similarity, and then choosing representative melodies from each cluster and building a Markov chain with them. A random walk approach can then be followed to generate a solo.

This algorithm produces reasonably good results: subjects were able to match solos generated by Impro-Visor with their original soloists whose corpora were employed in 85% of all cases. However, the algorithm lacks the "ability to develop its improvisations and to express emotions and intention", according to a study conducted in 2011 (Jordanous,

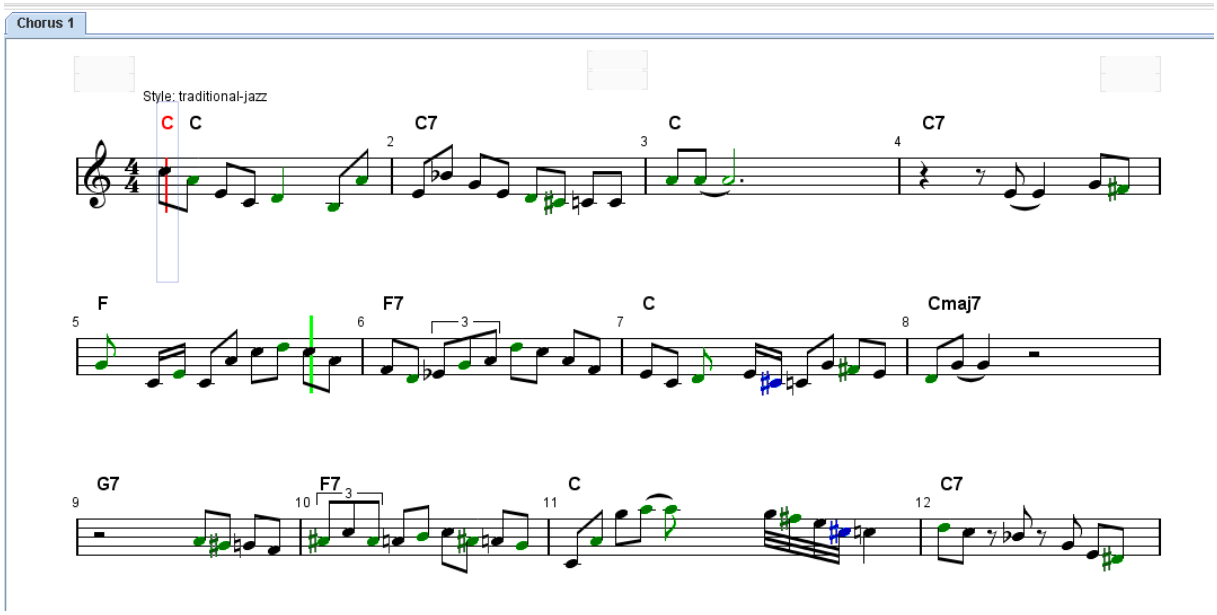


Figure 3.6: Example of a solo generated by Impro-Visor.

2012). In addition, the sheer abundance of features negatively impacts its usability, and hinders the interaction with the user.

The Impro-Visor team has recently directed their research towards the use of neural networks for music generation (Bickerman et al., 2010), with encouraging results.

3.3.2 Band-in-a-Box

Band-in-a-Box is a musicmaking software that focuses on the generation of backing tracks that correspond to a chord structure that can be inputted by the user or produced by Band-in-a-Box itself. The program offers a myriad of different styles for these tracks, and besides accompanying the user's playing, it can create both melodies and solos to fit the song.

The corpora for the solos can originate from two different sources: a MIDI representation or an audio recording (referred to as **RealTracks**). In both cases, the material is automatically adapted to fit the underlying chords.

Band-in-a-Box's main advantage is its ease of use: a novice user can generate complete instrumental backing tracks in a matter of seconds, and despite the high quantity of features, it is much more friendly than Impro-Visor. However, the corpora employed in the solo generation (especially in the case of RealTracks) are not deep enough and all solos from the same corpus tend to sound exceedingly similar. In addition, the user's role in the creation of songs is very limited.

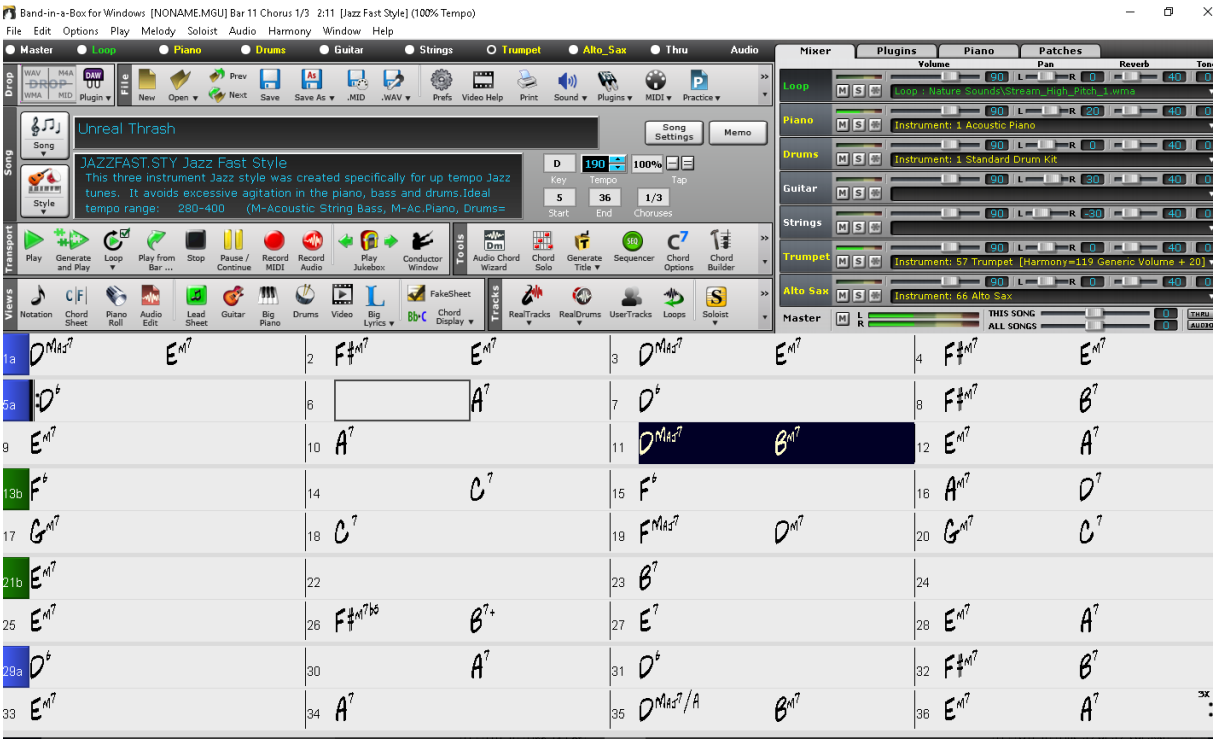


Figure 3.7: Band-in-a-Box main interface.

This software is updated yearly and new characteristics are introduced. Nevertheless, the underlying algorithm for the production of improvised output has not been made public.

3.3.3 GenJam

GenJam (**Genetic Jammer**) (Biles, 1994) is another improvisation system that employs genetic algorithm for the real-time generation of solos and is able to interact with a human player, adapting their solos and responding to them.

The GenJam algorithm uses a database of measures and phrases. Phrases consist of measures, which contain both pitch and rhythm information about a certain amount of notes. These notes are mapped onto actual pitches when chord progressions are incorporated.

When performing with a human, GenJam will listen to their solo and apply mutation operations to it before playing it back again as a response. These operations are simple ones, such as inversion, transposition or sorting. The best descendants can then be mutated again until the desired output is obtained.

Solos generated by GenJam are musically pleasing, and its ability to collaborate with

a human player closely coincides with how actual improvisation is created. However, it remains to be seen how similar the algorithm is to the thought process of a human improviser.

3.4 Quick review: why is it not enough?

The algorithms and programs that have been mentioned here have their own area of application, whether it is real-time accompaniment, teaching or musical composition, but they do not help us understand the cognitive problem of improvisation. The objective of nearly all of the mentioned examples is hence different to ours; for the most part, generating pleasant music. Nevertheless, a human improviser does not rotate and transpose the phrases of the other musicians, like GenJam does, nor does it cut and paste fragments of his or her corpus that fit into the current chord, like Band-in-a-Box.

It is true that some of the algorithms mention the "similarity to the original soloist or style" as a measure of quality, but similarity is only part of the problem. The novelty of the material produced is arguably more important: the ability of the algorithm to produce solos that are completely new but somewhat familiar. That is the aim of our algorithm, which will be discussed in detail in the next section.

Chapter 4

Algorithm

The core of the pattern-based algorithm had already been programmed in MATLAB. This code employs Markov chains to randomly generate both melodic and rhythmic patterns using probabilities extracted from the analysis of the corpus of jazz saxophonist Charlie Parker. The following sections go in depth into how this code operates, and expose its limitations. Following that, the new code that attempts to rectify and correct those restrictions will be examined in detail.

4.1 Previous versions of the code

Norgaard et al. (2013) contributes with an algorithm for the generation of tonal improvisations. Norgaard's paper is the basis for the present document, and it is this code that we have worked on and improved with the objective of removing some of its limitations.

4.1.1 Analysis of the corpus

The corpus used by Norgaard will also be employed here: it contains 46 solos by jazz saxophonist Charlie Parker. Every note in the corpus carries information about its length, pitch and position within the corpus. This is the data that will need to be transformed in order to generate a solo.

The concept of Markov chains appears here again. A Markov chain represents a process that satisfies the **Markov property**. This property states that the next state depends only on the current state. Speaking in mathematical terms, and considering the case where there is a finite number of states $X_1, ..., X_n$,

$$p(X_i|X_1, ..., X_{i-1}) = p(X_i|X_{i-1}),$$

where $P(\cdot|\cdot)$ denotes the conditional probability.

In our case, we will be using a Markov chain of order $k = 4$. We will call the order of the Markov chain our **interval length**. The interval length can be changed, but experimentation has shown that 4 is an optimal value: increasing it would yield fewer patterns and as a result less original material would be produced; decreasing it would skyrocket the amount of patterns and any resemblance with the corpus would be pure coincidence. The states of our Markov chain will be vectors of pitch or rhythm intervals: since the vectors will have length 4, they will store the intervals between 5 consecutive notes.

One could believe that this would violate the Markov property since the next step does depend on the k ones preceding it, but that is not the case: we can transform the process into an order 1 one just assigning a unique index to each vector and ignoring their size. However, we will continue to refer to it as an order-4 Markov chain because it contains information about the interval length.

To generate a sequence we will start by arbitrarily choosing the initial state, and then select between one of the adjacent states equally at random. In order to be able to do this, we first have to find what those states are going to be, and we must take both pitch and rhythm into account. For that purpose, we analyze the corpus to obtain both **pitch patterns** (intervals between consecutive notes in semitones) and **rhythm patterns** (duration and distance between notes), creating a list of those parameters for every 5 consecutive notes. This means we have interval vectors for both melodic and rhythmic aspects and, furthermore, they can be connected: every pitch vector is related to a set of rhythm vectors, and viceversa. This will be important because it allows us to create solos where the pitch and the rhythm are not independent from each other.

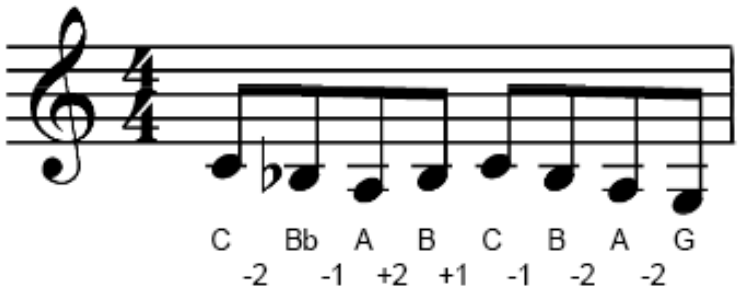
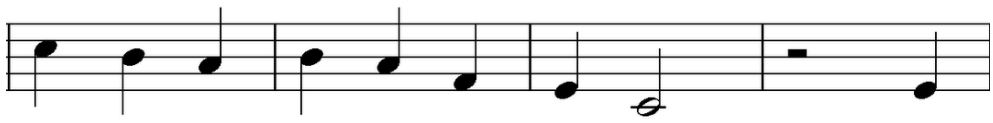


Figure 4.1: A sequence of 8 notes and the distance in semitones between them. Below, the four pitch patterns of length 4 that can be found in the sequence.

The rhythm analysis yields a matrix that contains information about the durations and the distances of notes within vectors of length 4: each row of this matrix is what

we call a rhythm pattern. Further operations are carried out on this matrix to remove repeated patterns, store their indices or count the number of occurrences of each pattern. This is one of the most time-consuming tasks for the algorithm, since we must loop and search over the entire rhythm analysis matrix, which has approximately 15,000 rows. Once this is completed, we have all the information about the rhythm that we need to start generating the solo.

The analysis that is performed on the pitches of the corpus is essentially the same: intervals are obtained subtracting pitches of consecutive notes, as long as those notes are not further away from each other than one measure and there are no rests between them. Then, these intervals are grouped in pitch patterns.



$$R = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 1 & 1 & 4 \\ 1 & 1 & 2 & 1 & 1 & 1 & 4 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} -1 & -2 & 2 & -2 \\ -2 & 2 & -2 & -4 \\ 2 & -2 & -4 & -1 \\ -2 & -4 & -1 & -4 \end{bmatrix}$$

Figure 4.2: Example of rhythm analysis and pitch analysis matrices for the sequence of notes above. In R, the first four columns represent the duration of notes, where a value of 1 is assigned to a quarter note. The last four columns are the distance to the next note: since there is a half rest after the half note, the distance to the next note is 4. In P, the pattern that would include the last note is not valid because of the appearance of the rest between the notes.

Chords must also be initialized: in this early version of the algorithm, the chord progression that is played during the solo is hardcoded and can only be modified within the actual code. To make matters worse, while the starting pitch of the solo is the tonic of the first chord of the progression, the rest of the notes are not affected by the adjacent harmony in the slightest, meaning that the solo and chords will be completely unrelated after the first few measures. It is clear why both of these solutions are a mistake, and

they will be corrected in the following version.

4.1.2 Solo generation

The probabilistic information that allows the existence of the Markov chain is stored in a **stochastic matrix**, which contains the transition probabilities between patterns. There is, as a result, a rhythm matrix and a pitch matrix. These are easily calculated using the indices of each pattern and the following one and increasing the value of the corresponding cell in the stochastic matrix, and then dividing by the total sum of appearances of the first pattern. This means the sum of each column of the stochastic matrix is equal to 1, with rows representing the current pattern and columns representing the next one.

$$S_R = \begin{bmatrix} 0.75 & 0 & 0 \\ 0.25 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

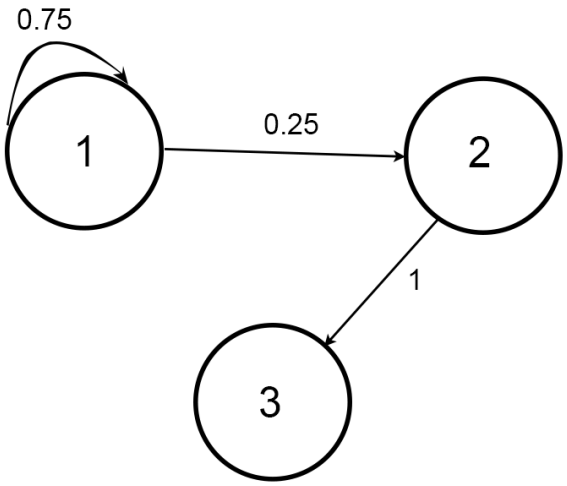


Figure 4.3: Rhythm stochastic matrix obtained from the sequence of notes in 4.2. There is a total of 3 unique rhythm patterns. The first column represents Pattern 1, repeated four times in the sequence: since it is followed by itself 3 times and by Pattern 2 once, the probabilities are the ones shown in the matrix. Pattern 2 is only followed by Pattern 3, and no continuation to Pattern 3 appears. Below, the corresponding Markov chain.

The next step is connecting rhythm and melody: this gives the algorithm the power to create solos where the durations and distances between notes are intrinsically linked to the pitches, which is one of its main strengths. With this purpose, we create two new data structures to collect the indices of the rhythm patterns that take place during a specific pitch pattern, and vice versa.

We are now ready to start generating notes: we have chosen a starting pitch at random from all the pitches with which a solo in the corpus begins, and the program also selects an arbitrary rhythm pattern and one of the possible pitch patterns for it given the aforementioned analysis. The user can input now the length of the solo in notes and the algorithm will produce them one by one.

The generation procedure is as follows:

1. Given the current pitch pattern P_0 and the current rhythm pattern R_0, we obtain the list of all the possible pitch patterns that can follow it and their associated rhythm patterns. We call this list of potential next rhythm patterns $R(P_0)$.

2. We randomly choose the next rhythm pattern $R_1 \in R(P_0)$, using the rhythm stochastic matrix to choose a rhythm pattern that can follow R_0.

3. We now obtain the list of all possible pitch patterns that can follow P_0 (using the pitch stochastic matrix) and are associated to the chosen R_1, $P(R_1)$.

4. We again select a random $P_1 \in P(R_1)$.

5. We repeat the process from step 1 using P_1, iterating until the solo is complete.

What results is a list of rhythm and pitch patterns that are translated to actual MIDI notes adding information about their onset (the onset of the previous note plus the distance between that note and the rest stored in the corresponding rhythm pattern), their pitch (the pitch of the previous note adding the interval of the corresponding pitch pattern) and their duration (explicitly in the rhythm pattern). An extra piece of information must be supplied by the user in order to determine some of these attributes: the tempo of the piece in beats per minute.

The structure of the solo is now complete. The version of the code being described now treats the concept of chords in a very rudimentary way: the progression is hardcoded by the programmer and then added to the solo with chords happening at the beginning of each measure, repeating itself until the solo is complete. No other adjustment is made in this version to make the solo fit the harmony.

Finally, a Java library handles the conversion of the data structure to a MIDI file that is playable by any standard music notation software such as Finale. MIDI (Musical Instrument Digital Interface) is a technology that allows the transmission of computerized musical information, and its ubiquity and compactness make it the best format for our output.

We have seen how the way the chords are treated (no interface for inputting them, not relevant in the generation of the notes) is the biggest problem of this algorithm. The next section will tackle our solution to these issues.

4.2 Our code

We have mentioned two issues related to the treatment of chords in the original code: they can only be modified within the code, and they do not affect the generated pitches. We will approach these problems one by one.

4.2.1 Interface

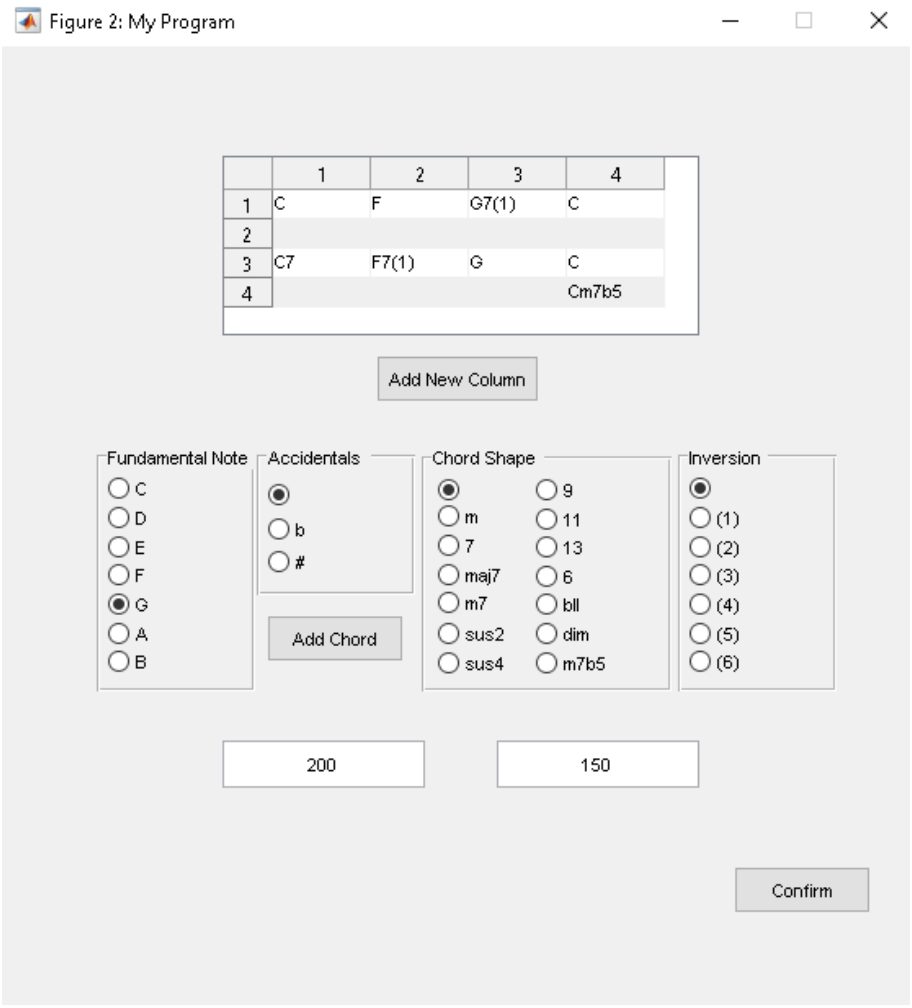


Figure 4.4: The program interface.

An interface was created using GUIDE, the user interface design package in MATLAB. Since MATLAB is mostly used for mathematical analysis, its GUI development enviroment possesses many shortcomings, particularly in the visual aspect. However, a

chord input interface is simple enough that there is no need for a more powerful design tool. In future versions, a different programming language could be used that produces more aesthetically pleasing results.

Chords are inserted in a grid (similar to a spreadsheet). Each column of the grid represents a measure, and each of the four rows is a beat within that measure. Chords can be inputted directly into a cell through the keyboard, or built with a radio button set. Thirteen different chord shapes are available.

In addition, text fields for the number of notes of the solo and the tempo of the piece have been included in the interface, instead of asking the user to input them via the command line.

A number of error messages can be displayed above the chord input grid. If the user tries to submit an empty chord grid, or one of the chords is invalid, or either of the text fields is empty or filled incorrectly, a red text will show explaining the error that was made.

4.2.2 Chord processing

The chord grid merely contains strings, which must be transformed into a sequence of pitches in order to add them to the final solo. This is achieved in two steps.

First, we build a list of chords that includes their duration. A chord will play at the beginning of every measure; if the corresponding cell of the grid is empty, then we will play the last chord that was inputted.

Once the list is complete, regular expressions are employed to check if the chord is correct and then find its fundamental note and chord shape. A fundamental pitch is assigned and modified if there are any accidentals (♭ or ♯). The code includes a data structure storing the semitones that have to be added to the fundamental note to build the rest of the chord: this is how the remaining pitches are obtained.

The user can also select chord inversions. The notation used, while not standard, makes it easy to rotate the members of the chord. Once that is done, the pitches are adjusted so they are arranged in ascending order.

The resulting list of chords will contain the duration and a vector with the pitches of each chord. We can now make a list of chord notes with the same information as the list of solo notes that was described earlier.

The final step of the process is making sure that chords and melody are related. The way we will guarantee that will be checking that every note that takes place during or immediately follows (by less than a measure, providing that there are no other chords in between) a chord is a member of that chord. If that is not the case, that note will be moved up or down in terms of pitch so it matches the closest member of the chord, and the rest of the solo will be transposed accordingly.

Going into further detail, the algorithm starts by finding the note that must be checked.

Issues of anticipation (when the note is played before the chord) and suspension (the playing of the note is delayed) must be taken into account, as well af those of precision. For example, a note is decided to occur at the same time as a chord if it starts within 1/32 times the size of a measure of it. For this reason, more than one note can take place on a specific chord when a combination of short notes and anticipation appears.

These notes are then examined to check if their pitch is part of the chord: if it is, no further action is required and we can move onto the next. If the note is not a member of the chord, we find the closest pitch to it that is, and transpose it up or down depending on the result. In order to preserve the original intervals that are part of the corpus, the rest of the solo is also transposed. A special case arises when a note occupies more than one chord: here we just consider the last of these chords to be the appropriate one and we transpose the solo accordingly.

Then, the next chord is found, and the first note that takes place during or after that chord is located. However, it is not safe to say that the chord we have is the one that this note is associated to, since it may be the case that there are chords in between when there is a large rest between two notes in the solo. For this reason, we must now find the actual chord that fits the note. Once we have the position of the next chord and the next note, we can repeat the process. When no more notes or chords are found, the procedure ends and the chords have been adjusted.

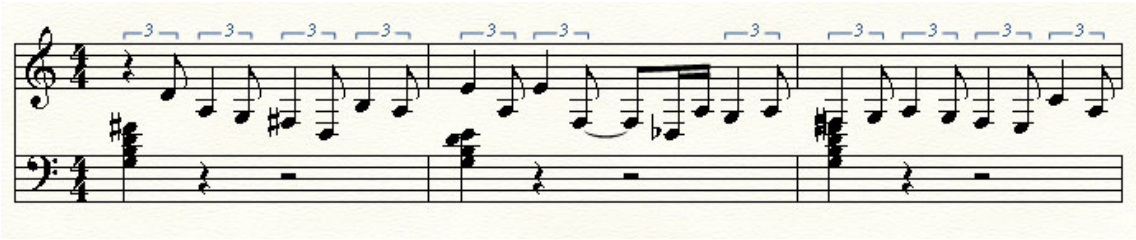


Figure 4.5: First three measures of a solo generated by the program, as seen on Finale.

Chapter 5

Tests

5.1 Validation tests

Testing is a fundamental task in every software development project. It aims to both ensure that it does what it should be doing and find defects before it is made available for users (Sommerville, 2010). The first of those goals is referred to as **validation testing**, while the second is known as **defect testing** or **verification**.

Since the inputs for testing must be artificial (not created by actual users, but the developer), it is of the utmost importance that they are designed carefully, putting thought into how the software is going to be used in a real environment.

Validation tests serve the purpose of checking that the software behaves according to the specifications. With this in mind a few test cases were created to assess the program's performance that are similar to how the software is going to operate in normal circumstances. Chord progressions from real jazz songs were employed so as to get results that are as close as possible to actual solos. Five solos with 125 or 250 notes were generated, with tempos ranging from 90 to 140 beats per minute. These solos can be found in the TestResults folder within the project and opened with any MIDI editing software such as Finale.

Defect testing was also done introducing incorrect data in the interface. Errors in the user input (empty chord grid, no solo length or BPM selected, incorrect chord) are automatically detected by the interface, which displays an error messages informing the user about how to solve the error. Other errors which had not been taken into account until this time, such as the ones related to inserting a negative input in either of the solo length or BPM fields, were fixed by limiting the range of inputs that the user can choose from: 20 to 2000 notes for the solo and tempos of 20 to 300 beats per minute.

5.2 Algorithm performance tests

Tests were conducted to discover how long it takes the algorithm to produce a solo and adjust that solo to the chords. 5.1 and 5.2 show the results: five samples were taken for each input size, and a progression with four different chords was used.

SOLO GENERATION								
Number of notes (n)	t_1	t_2	t_3	t_4	t_5	Average time spent ($\bar{t}$)	Standard deviation (σ)	Coefficient of variation (CV)
10	25.42	25.47	25.57	25.58	25.51	25.51	0.07	0.26%
100	25.56	25.68	25.76	25.73	25.67	25.68	0.08	0.30%
1000	25.99	25.71	25.65	25.69	25.73	25.76	0.14	0.53%
10000	27.84	28.83	27.74	27.49	28.10	28.00	0.51	1.83%
25000	30.80	31.39	30.67	30.62	31.25	30.95	0.35	1.13%
50000	37.86	37.87	36.12	37.05	37.04	37.19	0.72	1.95%
100000	52.40	52.22	59.02	53.37	54.80	54.36	2.80	5.15%

Table 5.1: Time spent in the solo generation (analysis of the corpus and note selection).

SOLO ADJUSTMENT TO CHORDS								
Number of notes (n)	t_1	t_2	t_3	t_4	t_5	Average time spent ($\bar{t}$)	Standard deviation (σ)	Coefficient of variation (CV)
10	0.09	0.08	0.08	0.08	0.08	0.08	0.004	5.45%
100	0.11	0.08	0.08	0.09	0.09	0.09	0.01	13.61%
1000	0.12	0.10	0.11	0.10	0.10	0.11	0.01	8.44%
10000	0.27	0.55	0.53	0.44	0.55	0.47	0.12	25.57%
25000	1.86	2.14	1.73	1.78	1.73	1.85	0.17	9.29%
50000	6.49	6.32	4.31	5.38	3.82	5.26	1.24	24.58%
100000	17.57	20.01	24.07	20.47	18.99	20.22	2.42	11.98%

Table 5.2: Time spent adjusting the solo to the user-inputted chords. The variation here is substantially higher. This could be owed to the fact that some of the solos might need more of an adjustment than others (due to more of their notes being members of their underlying chords).

A problem arose during the execution of this tests: when the input size was too large, it was more likely to choose a rhythm or pitch pattern that has no possible continuation, resulting in a failure. This error will not be reproduced when the software is operated normally, since the default solution is to stop the generation and continuing with the

process despite the solo length being shorter than what the user desired. An alternative would be starting over and never stop generating until the solo we generate matches the user-inputted length. This would negatively affect the performance of the algorithm, so no corrective measures were taken.

We can observe that the time spent finding the patterns in the corpus approximately equals 25 seconds. This is a prohibitely long amount of time especially when we are generating very short solos where the actual time spent generating the notes is almost negligible. A solution to this problem is easy to find and will be proposed in the next section.

Chapter 6

Conclusions

6.1 Evaluation of the algorithm

Amongst the algorithm's limitations, we can find its relative slowness, which is mostly owed to the time spent in analysis the corpus to find the patterns. It is technically not necessary to perform the analysis every time a solo has to be produced, so we can save a sizeable amount of time storing the patterns after extracting them for the first time. This solution is already partially implemented for the pitch portion of the algorithm: the MATLAB structure where the corpus is stored includes extra information such as a list of pitch intervals, a list of pitch patterns and the pitch stochastic matrix. Having calculated these data structures beforehand would save us a lot of time, although they still need to be indexed and related to their corresponding rhythm patterns, which would also need to be obtained.

The user interface also lacks intuitiveness and visual appeal, but its design was limited by the choice of programming language. The code could be translated to a language that possesses better GUI design interface libraries such as C++ or Python, but a trade-off must be reached to compensate for the advantages of using a math-oriented language like MATLAB.

Human judgments were not done due to the limited scope of this project. They would consist of tests conducted on jazz experts in order to determine whether the solos generated by our algorithm are actually similar to those of Parker. A questionnaire could be designed to ask relevant questions about the quality of the output, in order to compare it to other algorithms and perform statistical analysis.

6.2 Legal and social aspects

The topic of intellectual property is the most relevant one as far as this project is concerned; specifically, who is the owner of the pieces of music generated by our software. This

is a complicated question: computer-generated works are a relatively new phenomenon and Spanish law does not account for it yet (*Real Decreto Legislativo 1/1996, Ley de Propiedad Intelectual*, 2014). The generated output is obtained merely by random chance, without any direct intervention from the developer. It is true that the user does have the ability to constrain the output, but it mostly involves a transformation that happens after the solo has actually been generated and that completely preserves its structure.

A case could be made that Charlie Parker himself is the author of every solo that is generated using his corpus; after all, we are attempting to produce improvisations in his style. Spanish law however dictates that "the intellectual property rights of a work that is the result of a transformation correspond to the author of the latter", and since there is no natural person (one cannot say the author of the improvisation is the program itself) that is responsible for the work, we arrive at no conclusion. It can be expected that the blossoming of computer-generated works leads to a rigid legislative basis in the near future. However, it must be said that no matter who the author is, the work that is generated cannot be reproduced or distributed for commercial ends without permission.

Also important is the aspect of Norgaard et al. (2013), the work that our project builds upon, which also used Charlie Parker's corpus. This corpus of solos in MIDI format was transcribed by Peter Sprague and made available to Norgaard. Since our work is a continuation of Norgaard's, the corpus was carried over.

6.3 Future work

Besides correcting the limitations that were mentioned before, as well as conducting tests, several avenues are open for future development of the algorithm. One of the most promising ones involves merging both the motor pattern and rule-based approaches to the nature of improvisation to observe if the combination of the two could yield results that resemble actual improvisations.

A way this could be achieved relatively easily would be mixing Temperley's rule-based algorithm's output with the one generated by our algorithm. Two options have be considered:

- We select which algorithm is in charge of the generation with a user-controlled probability α, in a note-by-note basis.
- We produce one solo using each method, and then we make atomic operations (note insertion, substitution or deletion) to transform one into the other (this similarity metric is known as **Levenshtein distance**). A user-controlled value α regulates the extent to which the solo is modified.

Another functionality that could be included is letting the user choose whether the chord progression is looped for all the duration of the solo (the current state of the

algorithm) or not. With the former, the generated solo will almost certainly end in the middle of the progression, which is not ideal since it breaks the flow of the piece. The latter needs no input for the solo length, since it can generate notes until the chord progression is over, so it would probably be preferable from a user perspective, although the alternative allows for longer solos which are more useful for developing since we are more focused on having an algorithm that works correctly.

References

Bickerman, G., Bosley, S., Swire, P., & Keller, R. M. (2010). Learning to create jazz melodies using deep belief nets. In *Proceedings of the international conference on computational creativity* (pp. 228–237).

Biles, J. (1994). Genjam: A genetic algorithm for generating jazz solos. In *Proceedings of the international computer music conference* (pp. 131–137).

Gómez, F., Mora, J., Gómez, E., & Díaz-Báñez, J. M. (2016). Melodic contour and mid-level global features applied to the analysis of flamenco cantes. *Journal of New Music Research*, *45*(2), 145–159.

Johnson-Laird, P. N. (2002). How jazz musicians improvise. *Music Perception*, *19*(3), 415–442.

Jordanous, A. (2012). A standardised procedure for evaluating creative systems: Computational creativity evaluation based on what it is to be creative. *Cognitive Computing*, *4*(3), 246–279.

Keller, R. M. (2005). *Impro-visor leadsheet notation.*

Norgaard, M., Montiel, M., & Spencer, J. (2013). Chords not required: Incorporating horizontal and vertical aspects independently in a computer improvisation algorithm. In *Proceedings of the international symposium on performance science* (pp. 725–730).

Pachet, F. (2002). The continuator: Musical interaction with style. In *Proceedings of the international computer music conference* (pp. 211–218).

Pachet, F., & Roy, P. (2011). Markov constraints: steerable generation of markov sequences. *Constraints*, *16*(2), 148–172.

Real decreto legislativo 1/1996, ley de propiedad intelectual. (2014).

Rutherford-Johnson, T., Kennedy, M., & Kennedy, J. B. (2012). *The Oxford Dictionary of Music*. Oxford University Press.

Schmidt, R. A. (1975). A schema theory of discrete motor skill learning. *Psychological Review*, *82*(4), 225–260.

Sommerville, I. (2010). *Software engineering*. Addison-Wesley Publishing Company.

Temperley, D. (2007). *Music and probability*. MIT Press.

List of Figures

List of Tables

Index