POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Master of Science in Computer Engineering

The Virtual Jazz Player: real-time modelling of improvisational styles using Variable-Length Markov Models incorporating musicological rules

Supervisor:

Prof. Augusto Sarti

Master Graduation Thesis by:

Simone Bollini

Student Id. number 682508

Academic Year 2010-2011

POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Corso di Laurea Specialistica in Ingegneria Informatica

Il Musicista Jazz Virtuale: modellizzazione in tempo reale di stili improvvisativi con modelli di Markov a lunghezza variabile basati su regole musicali

Relatore:

Prof. Augusto Sarti

Tesi di laurea di:

Simone Bollini

Matr. 682508

Anno Accademico 2010-2011

Abstract

The statistical analysis and modelling of jazz improvisation has been the topic of several works in the field of computer science. In most cases the analysis is based on elementary musicological factors such as pitch, duration, intervals and so on. However when the analysis and the modelling concern a specific musical style and/or a particular process, as in the case of jazz improvisation, these elements turn out to be grossly inadequate for capturing the underlying generational process. In order to understand which way to go for modelling complex processes such as those that govern jazz improvisation, it is therefore important to understand how improvisation works in the first place. The idea at the base of this work is that the jazz musician improvisation capability relies on a musicological "model" progressively built upon rules and constraints. Therefore a rule-based generational model that reflects the musician's one seems to be the natural way to approach jazz analysis and modelling. The aim of this work is to explore this direction by building a *mode*-based generational model. This model relies on a knowledge base of *modes* (musical scales) used by jazz musicians. A jazz improvisation analysis tool is designed and implemented in order to realize a mode-independent representation of the sequence of notes played during the performance. This process can be viewed as a sort of orthogonalization of the melodic dimension from the mode dimension. This way the melodic patterns are reduced in number and complexity and can therefore be easily modelled. Using a Variable-Length Markov Model the patterns' dynamic is captured in a mode-independent way and a generational model able to perform an automatic continuation of the improvisation is obtained. The result is a *modal analysis* tool with a great didactic valence. Embedding this tool in a Markovian statistics modelling system we obtain a convincing improvisation emulator interesting both for didactic and music production applications.

Sommario

L'analisi statistica e la modellizzazione dell'improvvisazione jazz è un tema ricorrente nella ricerca nel settore della *computer music*. Solitamente l'analisi si basa su concetti musicali semplici come, ad esempio, l'altezza delle note, la loro durata e gli intervalli. Se però l'analisi e la modellizzazione sono circoscritti a un genere o uno stile musicale specifici, com'è il caso dell'improvvisazione jazz, questi elementi si rivelano inadeguati per definire e simulare il processo generativo che vi sottende. Per capire quale approccio seguire nel gestire un processo complesso come quello dell'improvvisazione jazzistica è fondamentale comprendere a fondo quale sia l'approccio dell'improvvisatore. L'abilità del jazzista nel costruire complesse traiettorie melodiche in tempo reale rispondendo a vari stimoli come, ad esempio, quelli armonici, è basata su un modello musicale che viene costruito e alimentato nel tempo con nuove regole e nuovi vincoli. Per questo motivo, l'applicazione di un processo generativo ispirato al modello musicale del musicista appare come l'approccio più naturale da adottare. L'obiettivo di questo lavoro è esplorare questa direzione tramite la progettazione e implementazione di un sistema dotato di un modello basato sui *modi* (scale musicali). Questo approccio prevede che il sistema incorpori conoscenze musicali relative alle scale utilizzate dai musicisti jazz. Occorre quindi prevedere lo sviluppo di uno strumento di *analisi modale preventiva* che consenta poi di rappresentare le linee melodiche e i pattern in maniera indipendente dal *modo*. Più specificamente, una volta trovato il modo musicale corrente, la traiettoria del pattern può essere specificata con una metrica valida "all'interno del modo", che non cambia al cambiare del modo stesso. Questa rappresentazione può essere vista come un'ortogonalizzazione tra la dimensione melodica e quella *modale*, che consente di semplificare enormemente la modellazione dei pattern riducendone la numerosità e la complessità, e nel contempo aumentandone l'occorrenza e la solidità statistica. Questo ci consente di costruire un modello di Markov a lunghezza variabile in grado di catturare la dinamica intrinseca dei pattern in maniera indipendente dal modo, e di trasformarlo in un modello generativo in grado di

implementare una “continuazione” automatica dell’improvvisazione nel rispetto di alcuni parametri stilistici del musicista che ha effettuato il training. Il risultato è uno strumento di analisi modale che, anche da solo, ha una valenza didattica significativa per l’insegnamento di molte metodologie improvvisative nel jazz contemporaneo. Questo strumento di analisi, completato con una modellazione statistica Markoviana si trasforma in un sistema di emulazione stilistica di notevole impatto percettivo, utilizzabile sia per applicazioni didattiche che di produzione musicale.

Contents

Chapter 1

Introduction

The statistical analysis and modelling of jazz improvisation has been the topic of several works in the field of computer science [9][10][11][14][15]. Many are the objectives and applications: understanding the improvisation process, determining metrics for evaluating style and creativity, developing didactic applications, etc. In most cases the analysis is based on elementary musicological factors such as pitch, note choice and duration, rests, intervals and so on. As these factors are shared by all western musical genres, it is on the parameters that describe them that we can compare and classify different styles. Nonetheless, if we go deep into a specific aspect of musical analysis and try to capture nuances and subtleties, we find that these elements often turn out to be insufficient. In fact, when the analysis and the modelling concern a specific musical style and/or a particular process, as in the case of jazz improvisation, pitches and intervals alone turn out to be grossly inadequate for capturing the underlying generational process.

Is the improvisational process completely free? Does the musician truly pick the notes of the improvised pattern from the whole range of notes that are available on the keyboard? The motivation for a new work in this direction lies on the following idea: when addressing a specific musical genre, the analysis and the model could be much more reliable if not only the results and measurements but also some information about the creation process are considered. Furthermore, an automatic “continuation” of the solo on the part of a machine could be more convincing and relevant to the style. This suggests us that the most suitable way to approach the problem is to add a musicological knowledge base and some information about the improvisator’s way of

thinking to the analysis and the modelling system. The interesting thing is that the jazz improviser's way of thinking turns out to be very precisely structured.

When people listen to jazz, they often believe that the soloists are "freely doing whatever comes." In fact, as experienced improvisers will tell, the soloist is rarely playing completely free. An improvisational soloist is always following a complicated set of rules and being creative within the context of those rules. Freedom in jazz improvisation comes from understanding structure and attaining from improvisational resources. The improvisation is often defined as instantaneous composition. In other words the soloist has a musical idea and plays it in the same time. In order to be able, musicians usually practice for hours every day in order to assimilate the structure and improvisational resources. In a first approach we can consider the structure of a tune as the harmonic progression and the resources as chords and scales. The harmonic progression is basically a series of chords: the choice of the types and the order of the chords determines the harmonic structure. The improvisational line has to follow this structure in a consistent way. Thinking to chords and scales when soloing helps reduce the complexity and know in advance which notes sound good. Typically the first stage in practicing a new tune is to consider the chord arpeggios and define which scales sounds well on every chord. This is called the chord/scale relationship and is one of the most popular approaches used in practicing and performing jazz improvisation. Essentially the chord and scale are different points of view of the same device: chords contain all the tones of a scale by arpeggiating to the 13^{th} [20]. This relation is applied to an important set of scales, well known by jazz performers: *modes* or *modal scales* [17][18][19][20]. The term "modes", in a modern sense, has the same meaning as "scales" and represents a common way to confer a specific sound and feeling to the solo. In fact, it is possible to improvise on a harmonic structure with different modes in order to obtain the desired effect.

If we consider an improvisation line without any particular musical knowledge we identify a note sequence belonging to the chromatic scale, i.e. the full range of available notes. The employment of lines with a coherent structure or the application of specific patterns by the musician is not pointed out by this representation. In fact, the intervals motion inside the chromatic scale is closely related to the underlying harmonic progression. This means that the sequence would probably not be effective on a different chord progression. One of the ideas at the base of this work is to interpret from an engineering point of view some musicological principles in order to find a representation of the note sequence structure independent from the harmonic progression. This can be seen as a sort of orthogonalization of the pattern dimension

from the melodic-harmonic dimension. One possible way to perform this separation is to create a model based on *modes*. This approach follows the way of thinking of almost all the improviser and presents many advantages.

First of all a performance analysis based on this kind of musical resources represents a useful didactic tool. The musician typically practices drills based on scales and arpeggios [19]: a software with the properties described above could record the performance, process the data during the session and provide a feedback to the user in real time. Furthermore an analysis based on modes can have several applications:

Editing application. The improviser usually builds melodic lines using specific resources (scales and arpeggios) compatible with the underlying harmonic structure. Since there is more than one scale compatible with every chord it can be interesting to hear how the solo sounds like if some modes are changed. This would be possible thanks to the mode independent representation and through a projection on other modes. An other possible editing application is to play back the entire sequence using a completely difference path in the modes space. This means that the not only the mode can be changed regardless of the chord type but also the mode temporal sequence can be changed.

Pattern analysis. During improvisation often musicians develop repetitive sequence of notes called pattern [21]. Except for very abstract and out-of-the-harmony lines, these patterns are often developed within a scale space. As far as the musical intervals are concerned the analysis of patterns is poorly effective while with the mode independent representation the pattern is easily detectable, as described in chapter 5. This allows analysing the patterns and seeing how they are built and linked.

Continuation. The pattern analysis described above can form the basis for an improvisation continuation. In fact, if the melodic lines are processed and made mode independent a system, in a learning phase, can gather statistical data on the melodic and pattern structure and then, in a running phase, use this information to produce a melodic line even on a completely new harmonic progression.

In conclusion the main objective of this work is to integrate the analysis and modelling tools used in music analysis with some a priori information on the improvisation process. The way improviser practice gives necessarily some information on their forma mentis. This information may be helpful in building didactic systems and develop more convincing continuation tools. In order to verify this convenience a *modal analysis* system has been implemented. This system records every note played a musician during an improvisation session and tries to identify which modal scales the musician is referring to. Since the scales can be changed in every moment the algorithm

performs segmentation. The improvised line is segmented in order to find the set of scales that fit at best the series of notes. The *segmentation* module works with the information provided by a *chord recognizer* and a *progression* module. These modules let the musician input a song chord progression and feed the *segmentation* module with the chord information on every measure. The results from these algorithms are presented with a graphic user interface built in the Max/MSP environment. The orthogonalization between patterns and modes provided by this module is used in two prototypes.

The first prototype is a tool for playback the performance changing the modes. More precisely this prototype is designed to store information about a harmonic progression, analyse an improvisation performance in terms of used resources, and store the performance with a mode independent representation. This first system version comes with some limitation and assumption. The structures, resources and melodic patterns used by jazz musician can be very complex. In this case a simple (but extensive) knowledge base of modal, pentatonic, hexatonic and octatonic scales is used [18]. The system accepts as input only MIDI instruments and leaves the audio input as further development. The modal analysis considers polyphonic input sources whereas the pattern recognition is based on a monophonic source. This means that in case, for example, of a keyboard or piano in input, both chords and phrasing are used in order to perform modal analysis. Another limitation concerns the time signature. In order to maintain the rhythmic complexity low only measures in 4/4 are considered.

The second prototype, after a learning phase, performs an improvisation continuation coherent with a real-time accompaniment. This module performs pattern recognition exploiting the modes. In fact, the detection of pattern in jazz licks, difficult if considering only intervals, is possible if the information about scales is taken into account. The core of this algorithm is an implementation of a Variable Length Markov Model (VLMM). Markov Chains are often used for statistical analysis in music using pitches, durations or other musical elements as states. Furthermore there are Markov Models of different orders, depending on the desired memory on past states. In this case distances relative to modal scales represent the states and the memory on the past intervals, called context, is of variable length. This prototype comes with the same limitation and assumption described above. Moreover In this prototype the model doesn't comprehend a statistical assessment and evaluation of the rhythmic patterns. What was done instead was a mere mimicking of the rhythmic patterns proposed in the learning phase. Although this simple solution can, in fact, offer surprisingly good results in a variety of situations, in any reasonably sophisticated jazz improvisation rhythm plays a crucial

role. While a comprehensive coverage and solution for the problem cannot reasonably be proposed within the scope of this thesis, we can begin unfolding the implications of studying the rhythmic evolution of a pattern by attempting a simple modelling based on a similar approach to that developed for melodic evolution within the mode.

Chapter 2 presents the basic musical elements used for the implementation of the knowledge base and the chord progression management. In Chapter 3, a system overview is presented. The main engine dedicated to modal analysis through segmentation is discussed in Chapter 4. Chapter 5 describes pattern analysis based on modal analysis. In Chapter 6 a rhythmic modelling approach is presented. Chapter 7 describes the implementation and testing of the modal analysis as main engine for a didactic tool and of the prototypes introduced above. Chapter 8 describes conclusions and future work. Appendix A is dedicated to the scientific background: here the analysis and modelling tools used in the system implementation are described.

Chapter 2

Musical background

One of the most popular jazz play-along books on the market [19] presents *Scales* and *Chords*, in addition to *Sounds* and *Silences*, as the basic ingredients in jazz music. The prove is that looking at any transcribed jazz solo from any era we would see much evidence of phrases which use scales, chords, diatonic patterns, chromatic passages, leaps and rests. Most improvisation in mainstream jazz is based on chord progressions. The chord progression is the sequence of chords that harmonizes the melody. Jazz players also refers to it as *changes*. The duration of a chord can be very variable. Usually it lasts a measure, sometimes two, sometimes only half or a quarter. A *fakebook* will give the symbol representing a particular chord above the corresponding point in the melody. Even more important than the actual chords, however, are the scales implied by those chords. An improviser, when playing over a specific chord, will normally play lines built from notes in compatible scales. In the next paragraphs the various chords and associated scales used in jazz are presented. This chapter will introduce the main aspects of the chord/scale relationships presenting some basic elements of music theory such as intervals, scales, chords type, tonality and so on. The arguments are addressed from a practical point of view. Please refer to the bibliography for a rigorous treatment.

1.1 Intervals

There are twelve different notes in traditional western music: C, C#/Db, D, D#/Eb, E, F, F#/Gb, G, G#/Ab, A, A#/Bb, and B. After the B comes the C an octave higher than the

first C, and this cycle continues. This sequence is called the chromatic scale. Each step in this scale is called a half step or semitone. An interval is a combination of two notes and is defined by the number of half steps between them [16]. The most common method to classify and name intervals is based on their quality (perfect, major, minor, etc.) and number (unison, second, third, etc.). Intervals with different names but spanning the same number of semitones may have the same width, provided that the instrument is tuned so that the 12 notes of the chromatic scale are equally spaced (a commonly used tuning system called equal temperament). However, they are defined by different notes. For example, a tritone is sometimes called an augmented fourth if the notes in the interval appears to describe a fourth. For example, the tritone interval from C to F# is called an augmented fourth, because the interval from C to F is a perfect fourth. Conversely, if the spelling of the notes in the interval appears to describe a fifth, then the tritone is sometimes called a diminished fifth. For example, the tritone interval from C to Gb, which is actually the same as the interval from C to F#, is called a diminished fifth, because the interval from C to G is a perfect fifth. In general, if any major or perfect interval is expanded by a half step by changing an accidental (the flat or sharp indication on the note), the resultant interval is called augmented, and if any minor or perfect interval is reduced by a half step by changing an accidental, the resultant interval is called diminished. It is also possible to have doubly diminished and doubly augmented intervals, but these are quite rare, as they occur only in chromatic contexts. The table shows the most widely used conventional names for the intervals between the notes of a chromatic scale.

Table 2.1 Musical intervals classification with the relative measures in halftones

N. of semit.	Diatonic interval	Short	Cromatic interval	Short
0	Perfect Unison	P1	Diminished second	d2
1	Minor second	m2	Augmented unison	A1
2	Major second	M2	Diminished third	d3
3	Minor third	m3	Augmented second	A2
4	Major third	M3	Diminished fourth	d4
5	Perfect fourth	P4	Augmented third	A3
6	Diminished fifth	d5		
	Augmented fourth	A4		
7	Perfect fifth	P5	Diminished sixth	d6
8	Minor sixth	m6	Augmented fifth	A5

9	Major sixth	M6	Diminished seventh	d7
10	Minor seventh	m7	Augmented sixth	A6
11	Major seventh	M7	Diminished octave	d8
12	Perfect octave	P8	Augmented seventh	A7

1.2 Major And Minor Scales

All scales are simply subsets of the chromatic scale. Most scales have 7 different notes (eptatonic scales), although some have 5, 6, or 8. The simplest scale is the C major scale, which is formed by the notes C, D, E, F, G, A and B. A major scale is defined by the intervals between these notes: W W H W W W H, where "W" indicates a whole step and "H" a half.

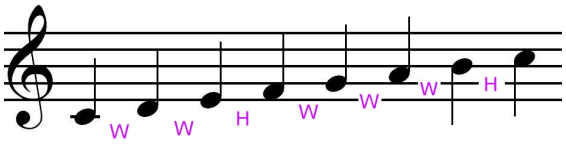


Figure 2.1 The C major scale. The intervals between the scale notes are expressed with W (whole tone) and H (half tone)

Thus, a G major scale is "G, A, B, C, D, E, F#", with a half step leading to the G that would start the next octave. The scale consisting of the same notes as the C major scale, but starting on A ("A, B, C, D, E, F, G") is the A minor scale. This is called the relative minor of C major, since it is a minor scale built from the same notes. The relative minor of any major scale is formed by playing the same notes starting on the sixth note of the major scale. Thus, the relative minor of G major is E minor. A piece that is based on a particular scale is said to be in the key of that scale. For instance, a piece based on the notes C, D, E, F, G, A, and B is said to be in the key of either C major or A minor. The chord progression of the piece may distinguish between the two. Similarly, a piece based on the notes G, A, B, C, D, E, and F# is either in G major or E minor. If the word "major" or "minor" is omitted, "major" is assumed. The collection of flat and sharp notes in a scale defines the key signature of the associated key. Thus, the key signature of G major is F#. Performers usually practice each scale in all twelve keys over the entire range of their instruments until they have complete mastery over all of them.

1.3 Chords

A chord is a set of notes, usually played at the same time, that form a particular harmonic relationship with each other. The most basic chord is the triad. A triad, as the name implies, is composed of three notes, separated by intervals of a third. For instance, the notes C, E, and G played together comprise a C major triad (Fig 2.2). It is so called because the three notes come from the beginning of the C major scale.

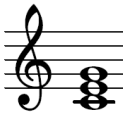


Figure 2.2 - C major chord formed by the notes C, E and G

The interval from C to E is a major third, and from E to G a minor third. These intervals define a major triad. A G major triad is composed of G, B, and D; other major triads are constructed similarly. The notes A, C, and E comprise an A minor triad, so called because the notes come from the beginning of the A minor scale. The interval from A to C is a minor third, and from C to E a major third. These intervals define a minor triad. The two other types of triads are the diminished triad and the augmented triad. A diminished triad is like a minor triad, but the major third on top is reduced to a minor third. Thus, an A diminished triad would be formed by changing the E in an A minor triad to an Eb. An augmented triad is like a major triad, but the minor third on top is increased to a major triad. Thus, a C augmented triad would be formed by changing the G in a C major triad to a G#. Note that a diminished triad can be formed from three notes of the major scale; for example, B, D, and F from C major. However, there are no naturally occurring augmented triads in the major or minor scales as illustrated in Fig 2.3.

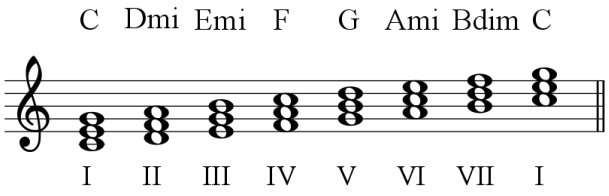


Figure 2.3 - Triads built on the C major scales

The harmony of popular music and jazz music is based on the major scale. Each of the twelve scales is a frame forming the harmonic sytem. However, chords of less than a

seventh are insufficient for jazz. If chords with four notes, separated by intervals of a third, are built on the diatonic scale we derive the scale-tone seventh chords (Fig. 2.4).

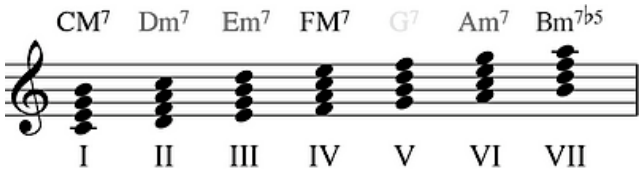


Figure 2.4 - Seventh chords built on the C major scale

The various combination of intervals in these chords form different values or qualities. On the I, and IV position we find the Major Seventh Chord; on the V position the Dominant Seventh Chord; on the II, III and VI position the Minor Seventh Chord; finally on the VII position there is the Half-diminished Seventh Chord. There is one chord used extensively in jazz harmony which does not appear naturally in any key: the diminished seventh chord. This chord may be formed at any point on the keyboard by building an interval combination of a series of minor thirds (Fig 2.5).

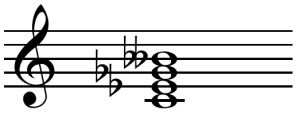


Figure 2.5 - The C diminished seventh chord

This relation between scale tone and seventh chord quality is true in any key. Jazz harmony basically utilizes five qualities and these can be applied in any point on the keyboard. There are twelve tones in the octave, each capable of supporting the five qualities. Thus, jazz harmony employs a Sixty Chord System [20].

Other interval combination are possible, although. A minor third can be added to an augmented triad, although this is a very rarely used chord that does not have a standard name in classical theory. Adding a major third to an augmented triad would create a seventh chord in name only, since added note is a duplicate an octave higher of the root (lowest note) of the chord. For example, C E G# C. Technically, the seventh is a B# instead of a C, but in modern tuning systems these are the same note. Two notes that have different names but the same pitch, like B# and C or F# and Gb, are called enharmonic.

More extensions to all types of seventh chords can be created by adding more thirds. For instance, the C major seventh chord (C E G B) can be extended into a C major ninth by adding D. These further extensions, and alterations formed by raising or

lowering them by a half step, are the trademarks of jazz harmony, and are discussed in sections below. While there is an almost infinite variety of possible chords, most chords commonly used in jazz can be classified as either major chords, minor chords, dominant chords, or half diminished chords. Fully diminished chords and augmented chords are used as well, but they are often used as substitutes for one of these four basic types of chords.

Three types of seventh chords have a very important relationship to each other: the major seventh, the minor seventh and the dominant seventh chord. In any major key, for example, C, the chord built on the second step of the scale is a minor seventh chord; the chord built on the fifth step of the scale is a dominant seventh chord; and the seventh chord built on the root of the scale, also called the tonic, is a major seventh chord. Roman numerals are often used to indicate scale degrees, with capital letters indicating major triads and their sevenths, and lower case letters indicating minor triads and their sevenths. The sequence Dm7 - G7 - Cmaj7 in the key of C can thus be represented as II-V-I. This is a very common chord progression in jazz. The motion of roots in this progression is upwards by perfect fourth, or, equivalently, downward by perfect fifth. This is one of the strongest resolutions in classical harmony as well.

1.4 The Circle Of Fifths

The interval of a perfect fifth is significant in many ways in music theory. Many people use a device called the circle of fifths to illustrate this significance. Picture a circle in which the circumference has been divided into twelve equal parts, much like the face of a clock. Put the letter C at the top of the circle, and then label the other points clockwise G, D, A, E, B, F#/Gb, C#/Db, G#/Ab, D#/Eb, A#/Bb, and F. The interval between any two adjacent notes is a perfect fifth. Note that each note of the chromatic scale is included exactly once in the circle. One application of the circle of fifths is in determining key signatures. The key of C major has no sharps or flats. As you move clockwise around the circle, each new key signature adds one sharp. For example, G major has one sharp (F#); D major has two (F# and C#); and so forth. Also note that the sharps added at each step themselves trace the circle of fifths, starting with F# (added in G major), then C# (in D), then G# (in A), and so forth. Conversely, if you trace the circle counterclockwise, the key signatures add flats. For example, F major has one flat (Bb); Bb major has two (Bb and Eb); and so forth. The flats added at each step also trace the circle of fifths, starting with Bb (added in F major), then Eb (in Bb), then Ab (in Eb), and so forth.

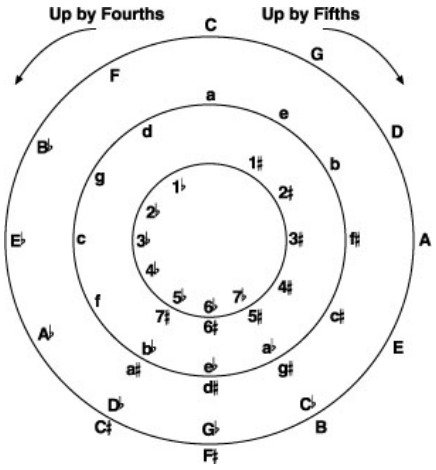


Figure 2.6 - Diagram representing the circle of fifths

The circle of fifths can also define scales. Any set of seven consecutive notes can be arranged to form a major scale. Any set of five consecutive notes can be arranged to form a pentatonic scale, which is discussed later. If the labels on the circle of fifths are considered as chord names, they show root movement downward by perfect fifth when read counter-clockwise. This root movement has already been observed to be one of the strongest resolutions there is, especially in the context of a II-V-I chord progression. For example, a II-V-I progression in F is Gm7 - C7 - F, and the names of these three chords can be read off the circle of fifths. One can also find the note a tritone away from a given note by simply looking diametrically across the circle. For example, a tritone away from G is Db, and these are directly across from each other.

1.5 Modal scales

1.5.1 Major scale modes

If we play the scale-tone chords in C and play the C major scale from root to root of each chord, we are playing the various modes of the scale of C. A mode is a displaced scale played from root to root of the chord.

Table 2.2 - Generation of the modes of the major scale

Chord	Scale	Displacement	Mode
I	C	C –C	Ionian
II	C	D – D	Dorian
III	C	E – E	Phrygian
IV	C	F – F	Lydian

V	C	G – G	Mixolydian
VI	C	A – A	Aeolian
VII	C	B - B	Locrian

These modes built on the twelve major scales represent one of the most important elements of jazz improvisation. They are highly effective in building a horizontal "blowing" line so long as the harmonic line moves in the normal scale-tone chords without alteration or chromatic adjustment. Of course this is rarely true. Even the simplest tune utilizes altered and chromatic chords. Though, this modal system must be expanded to meet the requirements of the Sixty Chord System. Anyway this is a fundamental set of scales that every jazz player practices in all the twelve keys.

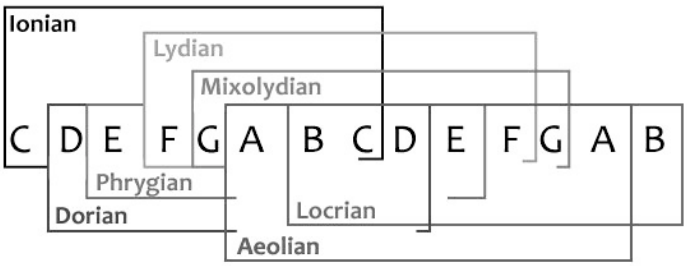


Figure 2.7 - The seven major scale modes

The major chord in any key appears on I and IV (Fig. 2.4). So the modal scale system offers two possible scale to be played on the major seventh chord. In determining which of these two modes to choose, the deciding factor must be the relative strength of these two major position in diatonic harmony. In the case of a modal tune the player must decide whether a natural 4th scale step (darker sound) or a raised 4th scale step (brighter sound) is desired. The dominant chord in any key appears on V only. In this case the dominant always takes the Mixolydian mode. Dominant chords on other than V are considered a temporary V of some other key. The minor chord appears on II, III and VI. In a chord series with a strong key feeling, for example I – VI – II – V- I, the three modes are used in their respective positions. In the case of a composition with non functional harmony the musician can choose the mode with the desired sound effect. The dorian is a more brilliant and opened sound scale, while the Phrygian is the darkest choice on a minor chord. The half-diminished chord appears on VII only. There is no doubt on the accompanying mode: this chord always takes the Locrian mode. The diminished chord has no natural position in any key. An arbitrary scale is employed for the diminished

chord which utilizes all the tones of the chord in addition to a series of chromatic tones. This set of scales is a very important material for jazz improvisation. The modes can be ordered from the brightest to the darkest scale:

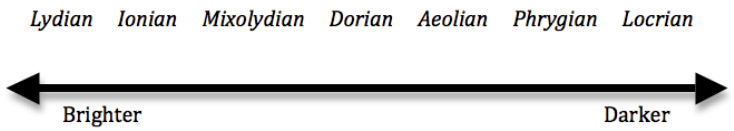


Figure 2.8 Major scale modes ordered from the brightest to the darkest

1.5.2 Ascending Melodic Minor scale modes

As the foundation of western music, the Major scale and its modes predominate in improvisation but other scale types still have an important role to play. For example, there are several chord types (such as augmented chords) which are not compatible with any Major scale mode. Also, the distinctive sound of the other scale types can add richness to the playing, even where Major scale modes are possible. The Ascending Melodic Minor scale is the next most useful in forming improvised melodies (Fig 2.9).

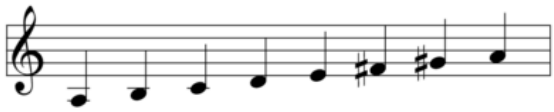


Figure 2.9 "A" Ascending Melodic Minor scale.

Like the Major scale, it can also appear in seven different modes, each starting on a different note. The same generation process used to find the modes of the major scale can be used on this scale. The following table shows the modes generated from the C Ascending Melodic Minor scale.

Table 2.3. - Modes generated from C Ascending Melodic Minor scale

Mode name	**Scale notes**
1. C melodic minor	C-D-Eb-F-G-A-B-C
2- D phrygian (w natural 6th)	D-Eb-F-G-A-B-C-D
3- Eb lydian augmented:	Eb-F-G-A-B-C-D-Eb
4- F lydian dominant:	F-G-A-B-C-D-Eb-F

5- G mixolydian (w b6th):	G-A-B-C-D-Eb-F-G
6- A aeolian (w b5)	A-B-C-D-Eb-F-G-A
7- B altered dominant:	B-C-D-Eb-F-G-A-B

1.5.3 Harmonic minor modes

The Harmonic Minor scale (Fig. 2.10) can also be used for improvisation, although less often than the Major or Melodic Minor scales, because its role in music is more for creating harmonies than melodies.

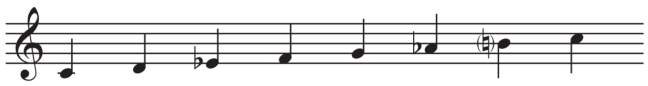


Figure 2.10 - C Harmonic Minor scale

As with the Melodic Minor, it can also be arranged in seven different modes, whose names are based on modified forms of other scales. The names of these modes are not standardised, so other names are possibly in use. The following table shows the modes generated from the C Harmonic Minor scale.

Table 2.4 - Modes generated from C Harmonic Minor scale

Mode name	**Scale notes**
C Harmonic Minor	C-D-Eb-F-G-Ab-B-C
D Locrian #6	D-Eb-F-G-Ab-B-C-D
Eb Harmonic Major	Eb-F-G-Ab-B-C-D-Eb
F Spanish Phrygian	F-G-Ab-B-C-D-Eb-F
G Double Harmonic Major	G-Ab-B-C-D-Eb-F-G
Ab Lydian b3	Ab-B-C-D-Eb-F-G-Ab
B Diminished	B-C-D-Eb-F-G-Ab-B

1.6 Other scale types

1.6.1 Pentatonic scales

A pentatonic scale is a musical scale with five notes per octave in contrast to a heptatonic scale such as the major scale and the relative modes. Pentatonic scales are very common in a number of musical style and genres. The ubiquity of pentatonic scales

can be attributed to the total lack of the most dissonant intervals between any pitches; there are neither any minor seconds (and therefore also no complementary major sevenths) nor any tritones. The two basic forms are the major pentatonic scale (Fig. 2.11) and the minor pentatonic scale (Fig. 2.12).

Figure 2.11 - C Major Pentatonic scale

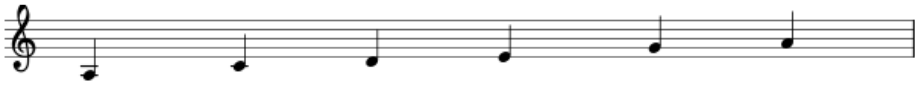


Figure 2.12 - C Minor Pentatonic scale

1.6.2 Octatonic scales

An octatonic scale is any eight-note scale. Among the most famous of these are two scale commonly used in jazz music: the Whole-Half scale (Fig. 2.13) and the Half-Whole scale (Fig. 2.14). The former is used on diminished chords while the latter is used on altered dominant chords.

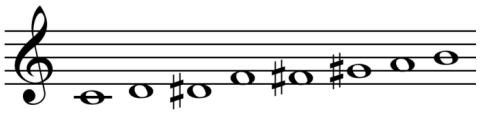


Figure 2.13 - Whole-Half tone scale

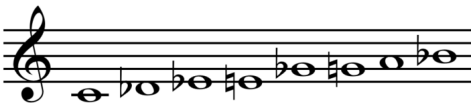


Figure 2.14 - Half-Whole tone scale

1.6.3 Whole tone scale

A whole tone scale is a scale in which each note is separated from its neighbours by the interval of a whole step. There are only two complementary whole tone scales, both six-note or hexatonic scales. Fig. 2.15 illustrate the whole tone scale on C. This scale is usually played on augmented chords. A vast number of jazz tunes, including many standards, use augmented chords and their corresponding scales as well, usually to create tension in turnarounds or as a substitute for a dominant seventh chord.

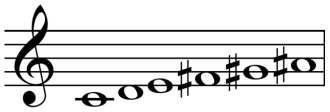


Figure 2.15 - Whole tone scale on C

Chapter 3
System overview

As seen in the previous chapters, the improvisation process is far from being a completely free and purely creative activity. On the contrary, it is often based on well defined and internalized structures and on the continuous practice of chords, scales, patterns, etc. A performance analysis tool based on the research of the devices described in Chapter 1 has many purposes. First of all it has a didactic utility: the musician that practices drills based on scales and arpeggios could have a feedback on his performance in real time. In order to practice improvisation typically the musician follows a process based on these steps: i) Selection of a tune, usually from the jazz repertoire and contained in a *fakebook*; ii) analysis of the chord progression in order to determine which modes/scales are adequate; iii) playing using the selected scales, arpeggios and building patterns on this scales. Sometimes artificial chord progressions are built in order to practice the same type of mode in different keys. In these cases the teacher checks the exercises or the musician performs self-assessment. A software tool that performs modal analysis during the improvising session could provide an interesting feedback for these types of exercise. Moreover such tool could provide analysis of transcribed solos in order to investigate the musical choices of a favourite musician. In fact, if a MIDI file with the improvisation line of the desired musician were analysed instead of processing the notes in input during a session it would be possible to see the choice of that musician in terms of modes in relation with the chords.

This type of analysis based on modes gives the opportunity to find a representation of improvisational lines independent from the harmonic progression. If

we express the lines in terms of notes or distances expressed in halftones we remain restricted to a specific harmonic progression. In fact, the note representation is an absolute reference and has no flexibility; the intervals in halftones can be transposed in every key but remain constrained by the chord type sequence. On the contrary if we express the melodic lines as distances on a certain mode we obtain a representation that is independent from the underlying chord. This can be seen as a sort of orthogonalization of the pattern dimension from the melodic-harmonic dimension. This encoded line can be applied on chord of whatever type or harmonic function: it is sufficient to choose the right mode. This method, described below in this chapter, can constitute the main engine for several interesting applications.

For example the information provided by the analysis can be used to edit a recorded improvisation session. The improviser usually builds melodic lines using specific resources compatible with the underlying harmonic structure. Since there is more than one scale compatible with every chord it can be interesting to hear how the solo sounds like if some modes are changed. This would be possible thanks to the mode independent representation and through a projection on other modes. An other possible editing application is to play back the entire sequence using a completely difference path in the modes space. This means that the not only the mode can be changed regardless of the chord type but also the mode temporal sequence can be changed.

Another application is pattern analysis. During the improvisation the musician has basically to possibility: he can either try to be very melodic and develop a lyrical line or work out a repetitive sequence of notes called pattern. In the latter case this sequence is not periodic in terms of absolute notes but has an internal coherence in terms of distances. In this case the word ‘distances’ instead of ‘intervals’ is used because usually the chromatic scale isn’t the only reference. In fact, except for very abstract and out-of-the-harmony lines, these pattern are often developed within a scale space. As far as the intervals are concerned the analysis of patterns is poorly effective:

Table 3.1 - Correspondence between notes and intervals

Phrase	*F*	*Ab*	*C*	*G*	*Bb*	*D*	*Ab*	*C*	*Eb*	*Bb*	*D*	*F*
Intervals (halftones)		+3	+4	-5	+3	+4	-6	+4	+3	-5	+4	+3

The sequence [+3,+4,-5,+3,+4,-6,+4,+3,-5,+4,+3] proceeds in a weakly repetitive way that is not useful in practice. Considering an Fm7 as the underlying chord the interval sequence could be used on every minor chord. However the operation is just a transposing and the pattern is not actually captured. Considering that the musician is

thinking of a Dorian scale it is possible to analyse the phrase in terms of distance between scale notes. For example, in the F Dorian scale (*F, G, Ab, Bb, C, D, Eb*) *Ab* is two step away from *F* and *Eb* is 4 steps away from *Ab*.

Table 3.2 - Correspondence between notes and distances relative to an F Dorian scale

Phrase	*F*	*Ab*	*C*	*G*	*Bb*	*D*	*Ab*	*C*	*Eb*	*Bb*	*D*	*F*
Distance		+2	+2	-3	+2	+2	-3	+2	+2	-3	+2	+2

Now the pattern is easily detectable. This encoding has many advantages: the phrase can be not only transposed but also continued, reduced, started from any point in the scale and also applied to other scales. In fact, if the pattern has an interesting construction and is effective from a musical point view it is likely effective also on other scales. This reflects the approach of improvisers that maintain the same pattern while changing the underlying scale. The listener perceives the continuation of the pattern on a new ground. This representation of the phrases in terms of note distances on a scale consists in a sort of *modal normalization* and can be used for an automatic improvisation.

The idea is to collect statistical information and build a model that represents the improviser way of building patterns (learning phase). Then this model is used to propose an effective melodic continuation. This continuation is supported by the *mode independent* representation and applies also on not yet visited chords, or even on entire new chord progression.

In this chapter an overview of the system is provided. The first part describes the design of a *mode analyser* and its possible stand-alone utilization as main component for didactic applications. Then some target application scenarios are considered for this type of analysis.

3.1 Modal analysis

The first part of this work is dedicated to the design of a performance analysis based on modes. The analysis is based on information collected during an improvisation performance. The objective is to detect which musical resources (*modes*) the improviser is using on every chord of a certain harmonic progression. Since the musician can change the *modes* at any time the analysis is based on notes segmentation in order to eventually detect a sequence of different modes. The segmentation is performed computing the accumulation of *outliers.* In this context a note is called an *outlier* in

relation to a specific scale if it doesn't belong to that scale. Therefore this type of segmentation requires a knowledge base, a sort of database for musical resources employed in the analysis.

In order to be used as a stand-alone didactic tool this analysis system has to be complemented with a system to interact with musicians. As described in the introduction of this chapter the jazz improviser usually practice playing different scales on specific harmonic progressions. Therefore such a system needs a notes input source (MIDI IN data) in order to capture the played note, data about the harmonic progression on which the user is going to play (input of chords type and duration), a time reference in order to follow the progression (metronome) and an efficient way to present the results about the *modal analysis*. The main information needed for a correct modes detection based on the described segmentation and analysis is illustrated in Fig. 3.1. This main system component receives the notes played by the improviser through a MIDI interface and other data relative to the chord progression and the metronome. The detected scale names are then visualized with a display system.

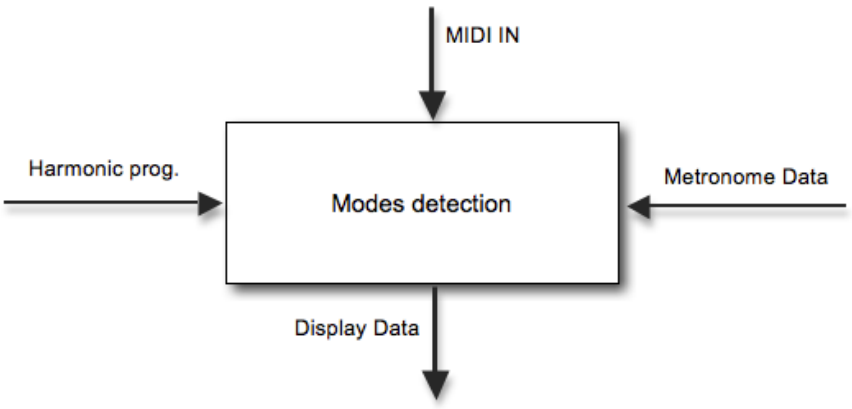


Figure 3.1 - Inputs and outputs of the modes detection module

The required functionalities of the system are designed in different modules interacting with each other and described in the following paragraphs.

3.1.1 Managing chord progression and metronome information

In the first stage the user has to input a chord progression. A *chord recognizer module* is required in order to receive the notes played simultaneously from a MIDI device and detect the five chord quality presented in Chapter 1 plus other chords with particular interval combinations. Then the user inserts the duration relative to each recognized

chord. This data is sent to the *progression module.* This module stores the sequence of chords and relative durations in order to present the progression graphically and supports the modules dedicated to the performance analysis and continuation. Since the precise location of musical events on the timeline (measures, divisions, subdivisions) is fundamental the user must set the desired velocity on the metronome module, the module dedicated to time management.

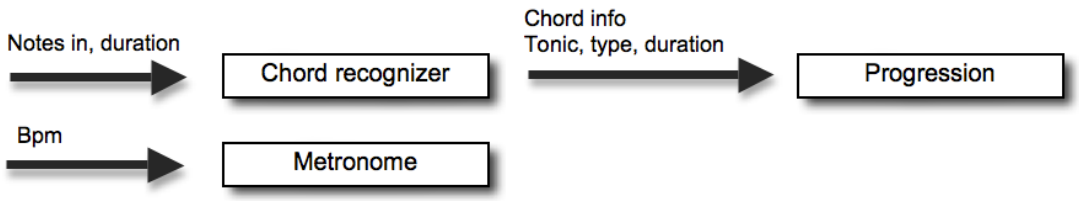


Figure 3.2 - Block diagram with the Chord Recognizer, the Metronome and the Progression module

3.1.2 Results visualization

The data stored in the progression module is sent to a component called *drawing module* dedicated to data presentation. The visualization is based on a simple chart where measures, divisions in quarter, chord names and qualities are displayed. The progression queries the *scales database* and adds some scale suggestions according to the inserted chords. Scales name are displayed with different colours following the sound line described in chapter 1. The connections between the describe module are illustrated in Fig 3.

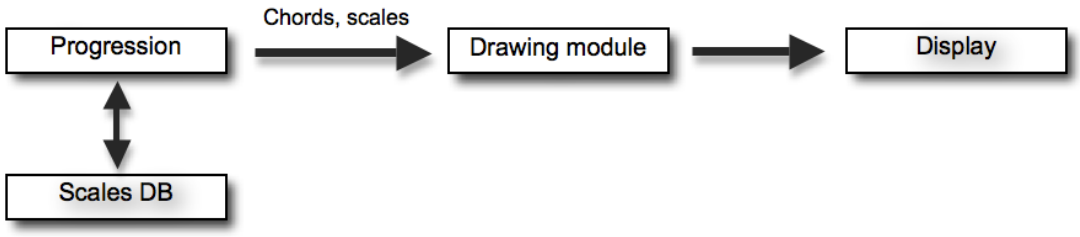


Figure 3.3 - Block diagram with the main connections between Progression module, Scales Database and Drawing module.

3.1.3 Modes detection

During the performance the user will hear the click from the metronome and a tracking system on the display will indicate the current point in the chord progression. In this way the player is aware of the current chord and the suggested scales in every moment. The module dedicated to modal analysis collects the notes. Basically this module is designed in order to detect modal scales used by the improviser on every chord. Since the duration of a single chord is variable and can extend to many measures the scale choice can vary too. Though, the analysis eventually detects a change of scale. This detection is based on segmentation of the notes in input: the set of scale that minimize the number of wrong notes (called outsider) is chosen.

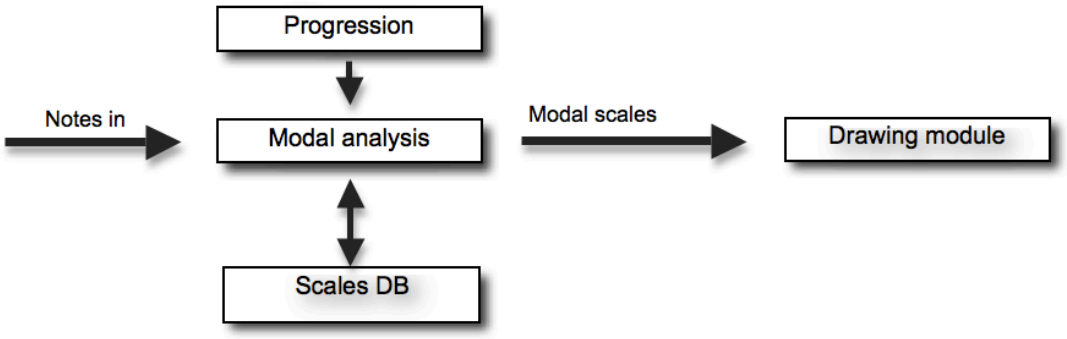


Figure 3.4 - Block diagram illustrating the main connections for the modal analysis module

This design of a *modal analyser* system provides a complete and independent tool because the results can be presented on a user graphic interface and are immediately useful as feedback on a jazz improvisation performance. However, this type of analysis can form the main engine for more complex applications. The next paragraphs present some interesting applications that are furthermore investigated in this work.

3.2 Target application scenarios

3.2.1 Changing path in the modes space

The improviser usually builds melodic lines using specific musical resources (scales and arpeggios) compatible with the underlying harmonic structure. Since there is more than one scale compatible with every chord it can be interesting to hear how the solo sounds like if some modes are changed. This would be possible using the results from the *modal analysis* to build a mode independent representation and project the processed line on other modes. This kind of application, addressed in detail in the next chapters, consists in the connection of the *modes detection* module with a simple performance recorder.

The behaviour of such a system can be described as follows. During a playing session, when the *modal analysis module* detects a modal scale a *recording module* computes the distances between the played notes in relation with the mode. These distances are stored in relation with the same rhythms and velocities used in the performance. Therefore it is a sort of recorder with a pattern/melody orthogonalization feature. The objective of this system is hearing the effect of changing the underlying mode of a melody or improvisational pattern. This application requires a **learning phase** in which the musician chooses a *mode* playing chords, performing an accompaniment or improvising. When the player considers that enough data has been provided to the system this is ready to pass to a ***running phase.*** In this phase the system performs a playback of the line played by the musician. Initially the mode-independent representation of the line is applied to the same mode used during recording. Then the user can determine the trajectory of the line in the modes space in real-time playing an accompaniment.

3.2.2 Pattern analysis and continuation

This application is based on the integration of the *modal analysis* with a pattern analysis and continuation system. The main idea is that pattern analysis would be much more effective if the information provided by the modes is used. In other words, instead of searching pattern in the chromatic scales using absolute intervals (number of halftones) the system computes the intervals on a *virtual modal keyboard*. First the pattern module records the note pitches. When the *modal analysis module* detects a modal scale the *pattern module* computes the distances between the played notes in relation with the modes. Then a model is built in order to gather statistical information about phrase structures. At this point the data inserted in the model is mode-independent. A block diagram is illustrated in Fig. 3.5 showing the integration of the Modal analysis module with a new system called Pattern module.

Figure 3.5 – Application of the modal analysis to a pattern analysis

Subsequently the model can be used to produce a continuation of the improvisation line that resembles the improviser style from a pattern point of view. This application could

verify that the mode independent representation not only is able to capture the pattern but also allows continuing the improvisation on completely new harmonic progression as far as the tonic, the chord types and the temporal sequence are concerned.

Chapter 4

Modal analysis

This chapter describes the methods, techniques and algorithms used in the design and implementation of a *modal analysis* tool. This type of analysis is based on a musical knowledge base and on an outliers-based segmentation. In the first section an overview of the *modal analysis* design is presented. Then the chapter continues with the description of how musical devices such as scales, modes and chords can be encoded in order to be processed by a computer program. The side modules that implement the requested functionalities are described in detail, too. Finally, the last part focuses on the main component that makes all the information from the other modules converge: the *modes detection* module.

4.1 Modal analysis design

As mentioned in the previous chapters *modal analysis* can be either a useful didactic tool or a component for further analysis applications. In case of a stand-alone utilization this tool requires additional functionalities in order to interact with a musician during a playing session. As shown in Fig. 1 the modal analysis requires an interface to acquire the notes played during session, information about the harmonic progression and data from a musical knowledge base.

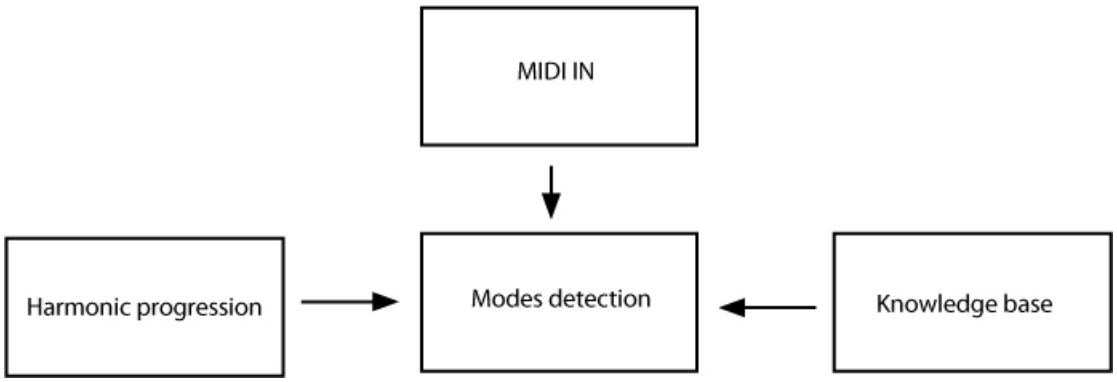


Figure 4.1 - Block diagram of the *Modes detection* module and the side modules required for a stand-alone didactic application.

The *mode detection* module can be viewed as composed by sub-functional blocks as illustrated in Fig. 4.2. The notes coming from the MIDI interface are collected in a buffer. The segmented unit is not the entire chord progression but is relative to the duration of a single chord. Therefore the buffer collects the notes during each chord, at the end of the chord the data is passed to the *Outliers processing* block. Here the outliers are computed and classified in relation with the resources present in the knowledge base.

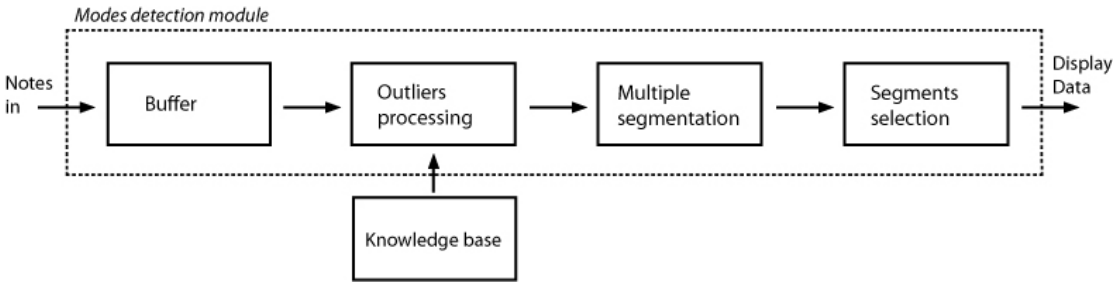


Figure 4.2 - Block diagram of the inner structure of the Modes detection module

All the data about the outliers is passed to the Segmentation block where multiple segmentation are performed depending on some parameters such as the *outliers tolerance* and *minimum scale length*. Finally, a cost function based on the number of outliers, the number of segments and their extension is used to determine the best segmentation.

4.2 Musical Resource management

The resource database is a system component consisting in a repository for the musical resources used by the improviser: modal scales, chord arpeggios, pentatonic scales and so on. This component provides information about scales to the main system

component: the *progression module*, the *modal analysis module* and the *pattern module*. Two classes named *Scales* and *Scale* for the scales management have been implemented. The *Scale* class is stored with the notes and some additional information such as the name and the colour used for displaying. The notes are encoded with an array of integers between 0 and 11 (the note range of an octave); every unit step represents the interval of a halftone. Each scale is stored one time in the key of C avoiding the transposition in every key as explained in the *progression module* paragraph. For example, the major scale has the following encoding:

Table 4.1 - C major scale encoding

Major scale	C	D	E	F	G	A	B
Encoding	0	2	4	5	7	9	11

The *Scales* class collects the various scale instances and implements some methods to access the data. Each scale is assigned to one of five specific collections representing the five chord qualities: for every chord type (major, minor, dominant, etc.) an Hashtable contains the compatible scale objects associated with the scale name as a key. Then different methods allow either to obtain the desired type of scales or the entire set of scales. For example, the method *getMinorScales()* returns the 5 choices for a minor chord: Melodic minor, Harmonic minor, Dorian, Aeolian, Phrygian.

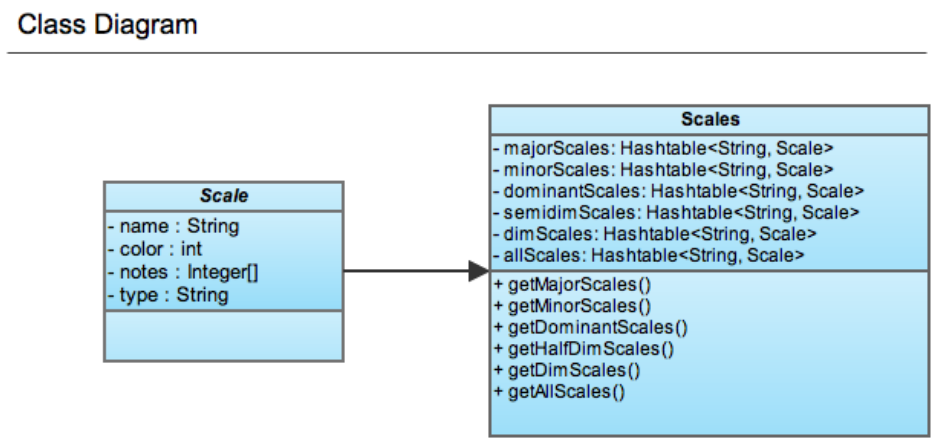


Figure 4.3 - Scale and Scales class diagram

The following table shows an example of musical resources present in the database with the relative encoding:

Table 4.2 - Scales encoding

Scale name	Encoding
Ionian	{0, 2, 4, 5, 7, 9, 11}
Dorian	{0, 2, 3, 5, 7, 9, 10}
Phrygian	{0, 1, 3, 5, 7, 8, 10}
Lydian	{0, 2, 4, 6, 7, 9, 11}
Mixolydian	{0, 2, 4, 5, 7, 9, 10}
Aeolian	{0, 2, 3, 5, 7, 8, 10}
Locrian	{0, 1, 3, 5, 6, 8, 10}
Locrian #2	{0, 2, 3, 5, 6, 8, 10}
Lydian b7	{0, 2, 4, 6, 7, 9, 10}
Superlocrian	{0, 1, 3, 4, 6, 8, 10}
Melodic Minor	{0, 2, 3, 5, 7, 9, 11}
Wholetone/Halftone	{0, 2, 3, 5, 6, 8, 9, 11}
Halftone/Wholetone	{0, 1, 3, 4, 6, 7, 9, 10}
Wholetone	{0, 2, 4, 6, 8, 10}

4.3 Chord recognizer

A chord recognizer has been implemented in order to speed up the chord insertion. This module allows the musician to play a chord on the MIDI controller. When the last key is released the recognizer starts the comparison within the chord in the database. The basic chord types are encoded with integers from 0 to 23 in order to capture the ornamental tone beyond the octave (9, 11, 13). The simplest chords are the triads:

Table 4.3 – Encoding of the four types of triad

Chord name	Encoding
Major triad	0, 4, 7
Minor triad	0, 3, 5
Augmented triad	0, 4, 8
Diminished triad	0, 3, 6

The encoding of the five type seventh chord requires an additional number:

Table 4.4 - Seventh chords encoding

Chord name	Encoding
Major seventh	0, 4, 7, 11
Minor seventh	0, 3, 7, 10
Dominant seventh	0, 4, 7, 10
Semi-diminished seventh	0, 3, 6, 10
Diminished seventh	0, 3, 6, 9

Other chords containing altered notes or additional tones beyond the seventh are brought back to one of the five basic types. For example:

Table 4.5 - Additional Seventh chords encoding

Chord name	Encoding
Dominant 7 b5	0, 4, 6, 11
Dominant 7 #5	0, 3, 8, 10
Sus 4-7	0, 5, 7, 10
Sus 4-7-9	0, 5, 7, 10, 14

The chord identification procedure requires playing the chord in the fundamental position, with the tonic as lowest note. The user can press the keys composing the chord in every position on the keyboard, simultaneously or at different time. When the last note is released the identification starts. The lowest note is taken as fundamental tone and is subtracted to all notes.

$$y_i = x_i - l \qquad i = 1, \ldots, N$$

where y_i are the notes in output, x_i the notes in input, l is the lowest note, N the number of played notes.

Table 4.6 - Chord notes encoding for identification

Input	66, 69, 73, 76
Encoding	0, 3, 7, 10
Identified chord	Minor seventh chord

4.4 Progression module

The progression module is a fundamental element in the system. It supports and coordinates the functioning of many other modules. As described in Chapter 2 the basic musical element on which the performance is based is the chord progression. The chords are an important framework for the improviser and affect the choice of arpeggios, scales, patterns, etc. Therefore, the information about the chord progression is of primary importance for the analysis of improvisation resources, pattern recognition and even more for a correct and coherent melodic continuation. Finally this module controls the graphic interface in order to give a real-time feedback to the performer, highlighting the actual point and chord in the progression.

The chord progression is loaded through the MIDI device using the chord recognizer module and the graphic interface. The data about every chord consists of the tonic, the chord type and the duration in quarters. The tonic is treated with a map between integers from 0 to 11 and the notes from C to B.

Table 4.7 – Tonic note encoding

Integers	Notes
0	C
1	C#/Db
2	D
.	.
.	.
.	.
10	A#/Bb
11	B

Every possible chord type is brought back to one of the five qualities described in Chapter 2. The following table shows the abbreviation used in the user interface for the five qualities.

Table 4.8 - Basic seventh chords name abbreviation

Abb.	Chord type
Maj	Major seventh chord
Min	Minor seventh chord
Dom	Dominant seventh chord
Semi	Semidiminished seventh chord
Dim	Diminished seventh chord

During the chord insertion the module updates the music chart with the chord names and suggests a set of scales for improvising. The scale names are displayed with different colours depending on the sound brilliance. When the performer switches on the metronome the module receives in input the beat (quarter) and starts going through the progression. On every beat the display is updated: a red growing line moves on the time axis and the actual chord is showed on a panel. When the current chord duration expires the module sends the information about the new one to the *analysis module* and to the *pattern module*. Furthermore, the tonic is used to transpose every incoming note in the key of C and beginning from 0. This simplifies the modal analysis: the handling of twelve different keys is avoided and the modal scales are stored only one time, in the key of C. The calculation for the transposition is:

$$y = [(12 - t) + x] \bmod 12$$

where t is the tonic and x the note number in input.
For example, suppose that the musician is playing four quarter notes in every measure. The data from the MIDI device looks like this:

Table 4.9 - MIDI Data management and encoding

Measure	**1**	**2**	**3**
Chord	Cmaj7	Fmaj7	Dm7
Tonic	0	5	2
MIDI notes input	60 – 62 – 64 – 65	65 – 67 – 69 - 70	62 – 64 – 65 - 67
Transposition	0 – 2 – 4 – 5	0 – 2 – 4 – 5	0 – 2 – 3 - 5

Comparing the transposed notes with the major scale (0,2,4,5,7,9,11) and minor scale (0, 2, 3, 5, 7, 8, 10) from the *scales database* it is clear that the notes played in the first and second measures belong to the major scale whereas the last measure is formed by notes from the minor scale.

4.5 Modal analysis

This system component, dedicated to modes detection, basically performs note segmentation. During the time dedicated to each chord of the harmonic progression this module receives a series of notes played by the improviser and organized in an array. The information about rhythm and relative position of the notes on the keyboard is

discarded because irrelevant to mode detection. For example, table 4.10 presents a sequence of notes played during a chord and the relative encoding for the mode detection:

Table 4.10 – Example of notes played on a chord

Current tonic	C (0)
Notes	C3, E3, G3, B3, C4, D4, E4
MIDI IN	60, 64, 67, 71, 72, 74, 76
Encoding	0, 4, 7, 11, 0, 2, 4

There are various possibilities: a) the sequence of notes played by the improviser is based on a particular scale choice; b) the sequence comes from more than one scale, this means that the improviser has changed scale during the same chord duration; c) the improviser based his improvisation line on one ore more scale using additional chromatic passages; d) the improviser plays an abstract or out-of-harmony line, no scales match that sequence of notes. These cases request a correct segmentation of the sequence of notes in order to eventually detect a change of scale during the same chord duration.

Table 4.11 – Example of encoding and segmentation of a sequence of notes played on a Dm7 chord.

Current chord	Dm7 Tonic: D (2)
Notes	D, E, F, G, A, B, C, Eb, D, F, Eb, G, F, A, Bb
Encoding	0, 2, 3, 5, 7, 9, 10 | 1, 0, 3, 1, 5, 3, 7, 8
Segmentation	0, 2, 3, 5, 7, 9, 10 -> Dorian 1, 0, 3, 1, 5, 3, 7, 8 -> Phrygian

The analysis and segmentation is performed computing the accumulation of *outliers.* A note is called an *outlier* in relation to a specific scale if it doesn't belong to that scale. For each scale present in the scale database it is possible to create an outliers array of the same length of the note sequence. If the note belongs to the considered scale the outliers array has a 0 in the same array position; the ones represents the outliers. In the following table a sequence played on a minor chord is presented. In this example three outliers array are showed relative to the Dorian scale, the Phrygian scale and the Whole tone scale.

Table 4.12 – Example of outliers computation for a sequence of notes played on a Dm7 chord.

Current chord	Dm7 Tonic: D (2)
Notes	D, E, F, G, A, B, C, Eb, D, F, Eb, G, F, A, Bb
Encoding	0, 2, 3, 5, 7, 9, 10, 1, 0, 3, 1, 5, 3, 7, 8
Outliers 1 (Dorian)	0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1
Outliers 2 (Phrygian)	0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0
Outliers 3 (H/W)	0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0

The outliers show that the Dorian scale fits good in the first part of the sequence, while in the second part presents 3 outliers. The outliers computed for the Phrygian scale presents an opposite situation. Finally the outliers relative to the Whole tone scale show that this choice is completely inadequate, having 10 outliers on 15 notes.

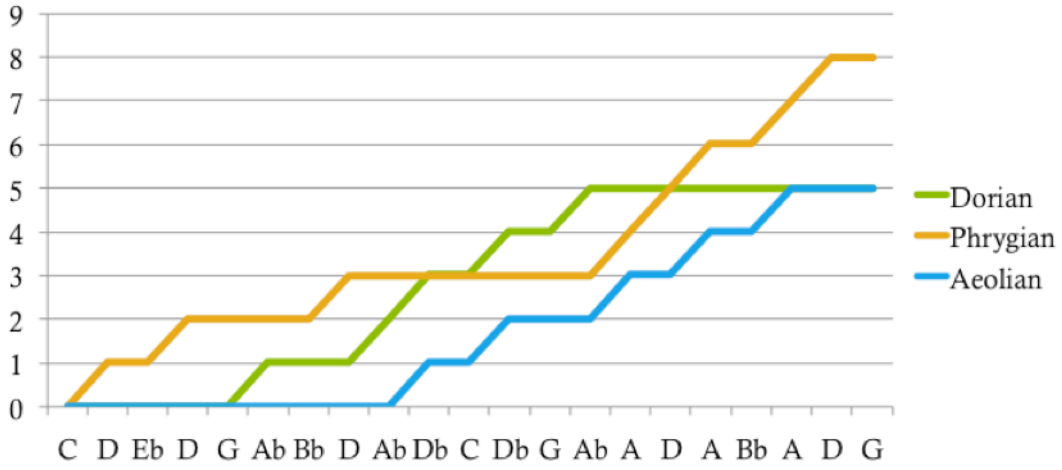


Figure 4.4 – Diagram with the accumulation of outliers. In this example the accumulation of outliers is computed for three different scale: the Dorian, the Phrygian and the Aeolian scale.

Table 4.13 – Example of segmentation based on three outliers arrays.

Notes	C, D, Eb, D, G, Ab, Bb, D, Ab, Db, C, Db, G, Ab, A, D, A, Bb, A, D, G
Outliers 1 (Dorian)	0 0 0 0 0 1 0 1 1 0 1 0 1 0 ~~0 0 0 0 0 0 0 0~~
Outliers 2 (Phrygian)	1 0 1 0 0 0 1 0 0 ~~0 0 0 0 0~~ 1 1 1 0 1 1 0
Outliers 3 (Aeolian)	~~0 0 0 0 0 0 0 0 0~~ 1 0 1 0 0 1 0 1 0 1 0 0

The segmentation procedure begins with the computation of the outliers arrays for the scales compatible with the current chord. Then two fundamental functions are used: the *extension* function and the *candidate* function.

The function *candidate* selects the first scale that fits the first notes of the input sequence. This function is conditioned by to parameters set by the user: the tolerance (TLR) and the minimum scale length (MSL). The former represents the level of tolerance of outliers above which the scale is rejected. The latter controls the level of segmentation: if a high MSL is requested the segments number will be low. Otherwise the note sequence could be highly fragmented with different scales. For every outliers arrays the function *candidate* computes the percentage of outliers. If the percentage remains under the tolerance threshold at least for MSL notes then the scale is selected. Then the function *extension* computes the number of notes for which the tolerance is respected. If at a certain point the scale is no longer adequate the function *candidate* is called again and an other scale is chosen. At the end of the notes sequence this algorithm has found a path through the outliers arrays. Since there is a tolerance on the outliers percentage this path could not be the best possible solution. In fact, the first compatible scale is selected but there may be other choices. Therefore, at every ramification point all the possible candidates should be explored and the relative outliers should be counted. This procedure resembles the visiting of different trees and is implemented with a recursive function called *minOutliers*. For every possible path through the outliers arrays the numbers of outliers is computed and finally the algorithm returns the segmentation that minimizes the following cost function:

$$C(o,s,e) = w_1 o + w_2 s + w_3 e$$

where *o* is the number of outliers, *s* is the number of segments, e is total segmentation extension (expressed in number of notes) and w_1, w_2, w_3 are the relative weights. The partial result is the set of all possible segmentations, accompanied by the scale name, the position and the number of outliers. A segment is formatted as follows:

[*starting position, end position, scale name, outliers number*]

For example, the next table shows a specific case for a measure with four compatible scales.

Table 4.14 - Four outliers arrays relative to four different scales.

Outliers 1 (Scale a)	0 0 0 1 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1 1 0
Outliers 2 (Scale b)	0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0

Outliers 3 (Scale c) 1 0 1 0 0 0 1 0 0 ~~0 0 0 0 0 0~~ 1 1 1 0 1 1 0

Outliers 4 (Scale d) ~~0 0 0 0 0 0 0 0 0~~ 1 0 1 0 0 0 0 1 0 0 0 0

In this case the *minOutliers* function returns the following data:

Segmentation 1:	[0, 6, a, 1][7, 13, c, 0][14, 20, b, 0]	n. outliers: 1
Segmentation 2:	[0, 6, a, 1][7, 13, c, 0][14, 20, d, 1]	n. outliers: 2
Segmentation 3	[0, 8, d, 0][9, 13, c, 0][14, 20, b, 0]	n. outliers: 0
Segmentation 4:	[0, 8, d, 0][9, 13, c, 0] [14, 20, d, 1]	n. outliers: 1

The segmentation n. 3 has the minimum outliers number and is therefore the best solution. The following pseudocode lines describe the main steps of the segmentation procedure.

```
function minOutliers is:
input: s position in the note sequence, outs outliers arrays
output: set of segments
1.      FOR each outliers array
2.          IF the array is a candidate for position s
3.              compute the extension from s
4.              update segments with the new segment
5.              IF end of sequence return segments
6.              ELSE minOutliers(s, outs)
7.          return
```

At the end of each chord duration, before the outliers arrays are computed, the algorithm selects the scales that will be used during the segmentation. This task is conditioned by an option set by the user through the graphic interface called "*use chord type knowledge*". If the option is *on* the system will select the scales from the database depending on the current chord type. For example, if the chord is D minor seventh only the minor scales are selected. Otherwise, if the option off, the system performs segmentation considering every scale in the database. In the first case the system gives for granted that the improviser will use the correct resources, without errors in the scale choice. The analysis will be more reliable if the assumptions are respected. It is likely that any chromatic approaches will be correctly managed. In the second case the musician has the freedom to make a chord type substitution and use an unexpected

scale. Using all the scale in the database the system will be able to detect this alteration but the risk of error is higher, too.

Using the chord type information in the scale selection gives the opportunity to discriminate between two types of *outliers*. Consider a major chord and two compatible scales: the Ionian scale and the Lydian scale (Table 4.14).

Table 4.15 In this table the difference between the Ionian scale and the Lydian scale is underlined. It consists in the fourth scale tone that in the case of the Lydian is sharpened.

Ionian	0 2 4 <u>5</u> 7 9 11
Lydian	0 2 4 <u>6</u> 7 9 11

The difference between these scales is a unique scale tone: the subdominant (IV tone). The sequence of notes played during the performance on this chord may present outliers that don't belong to the considering scale and neither to the other compatible scales. In this scale this outliers are 1, 3, 8, 10. This are notes not appearing neither in the Ionian nor in the Lydian scale. If the improviser is not substituting the chord function these notes are simply chromatic approaches and are called *outliers type A.* Otherwise, the outlier can indicate a possible change of scale. If the Ionian is considered and a raised fourth begin to appear this could mean a passage to a Lydian scale. This type of outliers is called *type B.* With this distinction the segmentation algorithm don't consider a foreign note in the outliers percentage if this note doesn't actually indicate a change of scale. If the option on the chord type is not selected this discrimination is not useful anymore because in a big set of scales there aren't outliers that don't belong to any scale.

The data produced by the modules described in this chapter is sufficient for the realization of a didactic tool. In fact, the information about the modes used during the performance and the correct positioning on a time axis could be an interesting feedback for jazz musicians. Anyway, with the next chapter we introduce a pattern analysis technique that benefits from the *modal analysis* results.

Chapter 5

Pattern analysis and continuation

5.1 Pattern analysis

The pattern analysis module is aimed at extracting the parameters of a predefined statistical model, which make the model "fit" the training excerpts at best. As described in Chapter 3, in order to better adapt to the musician's way of thinking, the statistical model works on specific musical scales that are adapted to the mode that has been previously detected and recognized. The pattern analysis module encodes an incoming sequence of notes in terms of *distances in the mode space,* as identified by the *modal analysis module.* Initially the module collects the notes played by the user as they come. These notes are processed to assign distances as soon as the mode is recognized by the mode analyser (this determines the underlying scale of that note sequence).

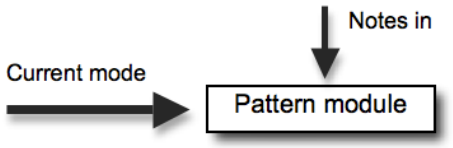


Figure 5.1 Pattern module inputs

During the playing session the notes are collected in a buffer. For example the players adds notes on a Em7:

Buffer: 64, 67, 66, 69, 67, 69, 71, 73, 74, 76

The note range (e.g. the keyboard) is split in two sections. The lower one is devoted to the accompaniment and the upper part for the soloing. The split point is chosen to be C3 (MIDI note 60), therefore the notes in the buffer are transposed using this formula:

$$y_i = x_i - t - 60$$

where x_i are the notes in the buffer and t is the tonic. In the example the tonic is E (encoded as 4) and the notes are transposed as follows:

Buffer: 0, 3, 2, 5, 3, 5, 7, 9, 10, 12

At the end of the chord the *modal analysis module* provides the used scale, a Dorian scale (Fig. 5.2), in this form:

Dorian scale: [0, 2, 3, 5, 7, 9, 10]

Figure 5.2 - The C Dorian scale

At this point the *pattern module* performs an extension of the Dorian scale notes in order to create a sort of "modal keyboard": a virtual keyboard where every key corresponds to a note of a specific mode. This model is used for indexing and computing the distances of the notes in the buffer. In this case the Dorian scale is extended as follows:

Mode: [0, 2, 3, 5, 7, 9, 10, 12, 14, 15, 17, 19, 21, 22, 24, 26, 27, 29, 31,
33, 34, 36, 38, 39, 41, 43, 45, 47]

The following pseudocode lines represent the main steps of the algorithm for calculating mode-based notes distances.

```
function distance is:
input: mode extension mode, notes buffer buffer
output: integers array distances
1.      note_old = first note in buffer
2.      FOR each note in buffer except the first
3.              position1 = index of note_old in mode
4.              position2 = index of note in mode
5.              add (position1- position2) to distances
6.              note_old = note
7.      return distances
```

The distance computation, as described above with the pseudocode, returns the following array:

Distances: +2, -1, +2, -1, +1, +1, +1, +1, +1

The pattern analysis module is based on the implementation of a Variable Length Markov Model. As described in Appendix A the Markov Model is an ideal tool for gathering statistical information in musical application and, in fact, is widely used in the computer music field. The main objective in the designing of this system part was to select the right tool in order to capture the patterns used by jazz players. The order of the Markov Model determines the memory size; for example, the first order has memory of the last state, the second order has memory of the two last states and so on. However, the pattern in jazz improvisation can be of variable length. The following table shows some examples.

Table 5.1 - Typical patterns used on scales

P1 Ascending scale notes	+1 +1 +1 +1 +1 +1 +1 +1
P2 Diatonic thirds based pattern	+2 -1 +2 -1 +2 -1 +2 -1
P3 Triads based pattern	+2 +2 -3 +2 +2 -3 +2 +2 -3
P4 Descending seventh chords pattern	-2 -2 -2 +5 -2 -2 -2 +5 -2 -2 -2 +5

Consider the following example of a C major scale played note by note in ascending direction from C3 to C4 as illustrated in Fig. 5.3.

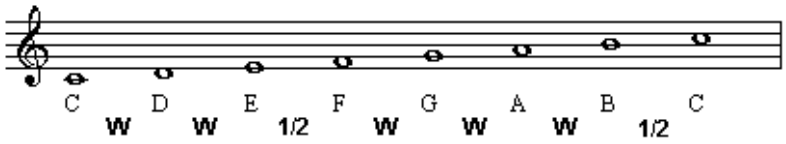
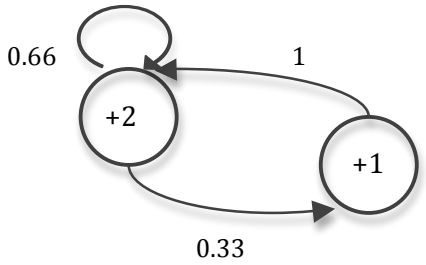


Figure 5.3 - The C major scale

If we compute the distances in halftones between the scale tones we obtain the following sequence:

$$[+2, +2, +1, +2, +2, +2, +1]$$

The *+1* steps are obtained because between *E* and *F* and between *B* and *C* there is only a halfstep as illustrated in Fig. 5.3. Using these distances as *states* for a first order Markov Model the following graph is obtained.



The associated transition matrix is:

Table 5.2 - Transition matrix for a first order Markov model modelling a C major scale

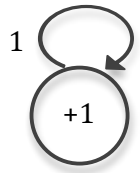
Context/next state	+2	+1
+2	0.66	0.33
+1	1	0

This model immediately points out a big limitation: the pattern on the major scale is not captured. If we want to use this statistics to make prediction and produce a continuation there is no assurance that the scale will be a major scale. Furthermore it will be not even sure that a scale starting from C will arrive to the C on the next octave. The scales will share the same statistics but every diatonic scale has the same statistics and this is not admittable. Using a Markov model of higher order the pattern is capture (Table 5.3), but the structure is not reusable on other scales and the model complexity increases.

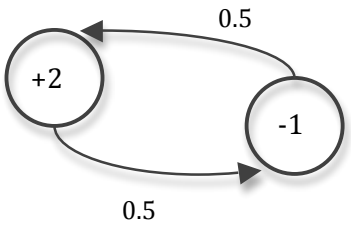
Table 5.3 - Transition matrix of a third order Markov model modelling a major scale

Context/next state	+2	+1
+2, +2, + 1	1	0
+2, +1, +2	1	0
+1, +2, +2	1	0
+2, +2, +2	0	1

If we use the mode-independent representation the scale played note by note in ascending direction forms the pattern P1 shown in Table 1. In this form the pattern has a periodicity equal to one step. Using each distance as a *state* a first order Markov Model could be enough to capture this ascending movement.



The P2 pattern represents an ascending motion on the scale with a jump forth and one step back (+2 -1). In this case the said model has the following graph.



This graph shows that the model is not able to capture the pattern. Since the probabilities are equal to 0.5 and the memory is 1 after the sequence (*+2, -1*) can follow with the same probability a *+2* or a *-1*. To properly catch the pattern a second order Markov Model is requested, as shown by the following transition matrix:

Table 5.4 - Transition matrix for a Markov chain with order 2

Context/next state	+2	-1
(*+2, -1*)	1	0
(*-1, +2*)	0	1
(*+2, +2*)	0	0
(*-1, +2*)	0	0

The P3 pattern is periodic every 3 steps. Though, a context length equal to 3 is required to properly model the pattern:

Table 5.5 - Transition matrix for a Markov chain with order 3

Context/next state	+2	-3
(*+2, +2, -3*)	1	0
(*+2, -3, +2*)	1	0
(*-3, +2, +2*)	0	1
(*-3, -3, +2*)	0	0
(+2, *-3, -3,*)	0	0
…	0	0

In order to avoid any constraint on the pattern length and maintain the maximum flexibility on the context length a Variable Order Markov Model was chosen as the right tool for this application. For a detailed description of the Markov Models please refer to Appendix A.

The implementation of the Variable Length Markov Model used in this application is based on a prediction algorithm called *Probabilistic Suffix Trees* (PSTs). The implementation of this algorithm, called *VMMPredictor* and described in Appendix A, provides 3 basic methods:

1. *init(int abSize, int vmmOrder):* Initialization with the alphabet size and model order.
2. *learn(java.lang.CharSequence trainingSequence): VMMPredictor* use trainingSequence and constructs its model.

3. *predict(int symbol, java.lang.CharSequence context):* Predicts the next symbol according to some context.

The distances array computed when the *mode* is provided forms the learning string for the *VMMPredictor*. The algorithm accepts a char sequence as learning string. Therefore the distances are encoded with symbols. The considered distances start from a descending interval of 12 steps on the scale to an ascending interval of 12 steps on the scale. A distance over this range is considered as the starting point for a new pattern and therefore is not encoded as distance.

Table 5.6 - Encoding of distances on the scale

Negative dist.	Symbol	Positive dist.	Symbol
-12	a	0	m
-11	b	1	n
-10	c	2	o
-9	d	3	p
-8	e	4	q
-7	f	5	r
-6	g	6	S
-5	h	7	t
-4	i	8	u
-3	j	9	v
-2	k	10	w
-1	l	11	x
		12	y

For example the pattern relative to the descending arpeggio of seventh chords on a scale (-2 -2 -2 +5 -2 -2 -2 +5 -2 -2 -2 +5) is encoded with the following string: **kkkrkkkrkkkr.** Then the *learn* function of the PST is called passing the learning string as parameter. The *VMMPredictor* constructs its model and becomes ready to make predictions.

5.2 Continuation

When the learning phase is over and the model is ready the system is able to pass to the running phase. In the running phase the module stops to receive new notes from the keyboard and propose a continuation of the improvisation. In this phase the VLMM model has learned the statistic about patterns with a mode independent representation. Therefore the system can produce new patterns without knowing in advance the modes on which the patterns will apply. The continuation process is based on the following steps:

1. Model construction based on a learning string
2. Selection of the initial context
3. Prediction of new patterns and bufferization
4. Setting of the current mode
5. Determination of the next note in relation with the current mode

5.2.1 Model construction

At the end of the learning phase the VMM Predictor constructs the model on the basis of the learning string encoded as described above. The running phase can be followed by another learning phase establishing a sort of interplay between the user and the machine. In this case two different policies are possible depending on the application at hand. In the first case the user may want a new learning phase in order to try something completely new and hear how the system reacts. In this case the learning string is reset and the effective notes for the next continuation are relative only to the last learning phase. In the second case the user may want to establish a sort of dialogue with the machine providing new information with alternated learning phases. The user expects to hear how the system reacts as he gradually accumulates knowledge through multiple learning phases. In this case the model can't be upgraded with the new information. A new model has to be constructed using a cumulative learning string.

5.2.2 Context selection

The prediction of new patterns for the continuation of the improvisation is based on a context. The context at the moment of prediction of the first distance can be only based on the sequence of notes played by the user. The context length can vary depending on the desired effect: with a long context or even with all the notes history the prediction will be more linked with the past sequence and will have a low surprise effects. Otherwise with a short context or even an empty string the first prediction will be less bounded and the probability of an interesting mixing of the past patterns will be higher.

5.2.3 Prediction of new patterns

The production of new melodic lines and/or patterns consists of the prediction of new distances based on a context. In fact, multiple subsequent predictions provide similar patterns from a statistical point of view. The prediction is based only on *visited states* in the Variable Length Markov Model. In other words predictions for distances not present in the learning string and hence with probability equal to zero are not requested. For example consider the following sequence of distances:

[-2, -3, +4, -2, -3, +4, -2, -3, +4]

The distances are encoded and the resulting learning string is:

[*k, j, q, k, j, q, k, j, q*]

Only the prediction for *k*, *j* and *q* are requested. For example, using an hypothetical context *C* we could obtain the following probabilities:

Pred("k", C) = 0.25
Pred("j", C) = 0.25
Pred("q", C) = 0.50

***k* 0.25**	***j* 0.25**	***q* 0.50**

Obviously, the sum of the probabilities of every visited distance is equal to one. When the distribution is defined a random number between zero and one is generated and the correspondent symbol is selected. The symbol is decoded and inserted in a buffer. The process described in this paragraph is repeated a number of times in order to fill the buffer. The buffer can be sized depending on the application at hand. The distances in the buffer are ready to be applied to any mode.

5.2.4 Setting of the current mode

The current mode can be set in any time in order to make the continuation coherent with an underlying harmonic progression. For example the change of mode can be synchronized with the progression module described in the last chapter. In fact, the progression is able to suggest a compatible mode in correspondence with a change of chord. Otherwise, the user can set a new mode freely performing an accompaniment to the automatic continuation as implemented in the prototype described in Chapter 6. The information is provided in the same form used for storing the scale in the knowledge base:

Ionian scale: {0, 2, 4, 5, 7, 9, 11}

This set of integers representing the scale is extended in order to cover a sufficient octave range for the notes in output. This operation, once again, consists in the creation

of a sort of modal keyboard. The array is transposed in the right key and extended for five octaves in the upper part of the keyboard. For example, in the case of an E tonic (4) the result an array already containing the MIDI notes:

Current mode array: [48, 50, 52, 53, 55, 57, 59, 60, 62, 64, 65, 67, 69, 71, 72, 74, 76, 77, 79, 81, 83, 84, 86, 88, 89, 91, 93, 95, 96, 98, 100, 101, 103, 105, 107]

5.2.5 Determination of the next note

Once the current mode has been set and the *current mode* array has been computed the system is ready to choose the real notes that will be played. The system uses an index called *mode_position* in order to move up and down in the *current mode* array. The starting value of the index is chosen as the last note played by the musician. Then the first distance is read from the *distances array* and is algebraically added to the *mode_position* index. This way the pattern computed in the previous steps is applied to the current scale.

The passage between modes with a great tonic difference (for example, greater than one tone) can result in a discontinuity in the improvisation line in output. For example, consider a simple pattern consisting in an ascending motion on a scale: [+1, +1, +1, +1, +1]. Initially this pattern is applied to a C major scale and suddenly the mode changes to a F major scale.

C major scale mode array: [..., 60, 62, 64, 65, 67, 69, 71, 72, 74, 76, 77, 79, 81, 83, ...]
F major scale mode array: [..., 65, 67, 69, 70, 72, 74, 76, 77, 79, 81, 82, 84, 86, 88, ...]

The two arrays show that if the *mode_position* index is incremented by one the user perceives an ascending motion on the scale. However, when the mode is changed the incremented index on the new array involves a jump, for example, from note 69 to note 76. To avoid this pattern interruption the modes arrays are not built starting from the tonic but are aligned in order to minimize the difference of notes at the same index.

This chapter proposed an approach to pattern analysis based on a mode-independent representation model. The massive employment of patterns by jazz musician during an improvisation can be easily captured with the presented model. In fact, due to the non-linear structure of the diatonic scales and of other musical resources the patterns

remain hidden in the chromatic scale. On the contrary, with a mode-independent representation the periodicity of jazz patterns are highlighted. Furthermore using Variable Length Markov Models the memory on past states (formed by distances on the modes) can be varied as needed.

Chapter 6

Approaching rhythmic modelling

In the previous Sections, the modelling of jazz improvisation did not consider a statistical assessment and evaluation of the rhythmic patterns. What was done instead was a mere mimicking of the rhythmic patterns proposed in the training sequences. Although this simple solution can, in fact, offer surprisingly good results in a variety of situations, in any reasonably sophisticated jazz improvisation rhythm plays a crucial role. The rhythmic dimension offers a huge number of possibilities in building a phrase. The variables are numerous and even two slightly different rhythmic phrases can arouse a completely different effect. Anyway, also in this dimension a jazz musician rarely acts in a totally free way. Usually the rhythmic structure of a phrase has some degree of coherence with the previous phrases and is based on patterns, too. While a comprehensive coverage and solution for the problem cannot reasonably be proposed within the scope of this thesis, we can begin unfolding the implications of studying the rhythmic evolution of a pattern by attempting a simple modelling based on a similar approach to that developed for melodic evolution within the mode.

The rhythmic pattern is generally not independent of the melodic pattern. However, often the jazz improvisator capitalizes on a "forced independence" between them as an interesting artistic tool. Furthermore, assuming independence between rhythmic and tonal patterns, indeed simplifies the modelling a great deal, with fairly acceptable consequences on the result. This is why we chose to do so, at least for this initial assessment. Therefore an independent Variable-Length Markov Model is used to

capture rhythmic patterns. Then predictions based on that statistical model are used to compute future rhythmic pattern that are associated to the melodic patterns.

6.1 Overview

The rhythm modelling can be viewed as a process formed by a sequence of operations as illustrated by the block diagram in Fig. 6.1 . The data coming from the MIDI interface is used to capture rhythmic events and convert them in a series of notes durations and rests. This data is quantized in order to reduce complexity, discard too short durations and remove other eventual inaccuracies in the performance. When the buffer of quantized notes reaches a certain threshold (that can be, for example, the length of a measure or the duration of each chord) the buffer content is passed to the "Segmentation and encoding" block. Here the rhythm information is segmented in parts with a duration equal or multiple of a beat. These parts are used as *states* for a Markov model. Associating a symbol to each state a learning string is produced and used for building a new VLMM instance. This model is used to make predictions on future states using a variable length context (please refer to Appendix A for a detailed description of VLMMs) that can be linked to the context used for melodic patterns. The predicted states are the decoded and made available in an output buffer.

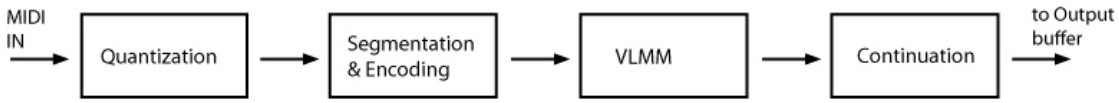


Figure 6.1 Block diagram of the rhythm modelling process. First notes durations and rests are detected and quantized. Then the rhythmic figure is segmented and encoded in order to prepare a learning string for the VLMM. Finally, the system performs model-based continuation.

6.2 Quantization

The rhythmic data filtered from the MIDI signal is the *duration* time and *delta* time. The *duration* represents the elapsed time in milliseconds between a NOTE ON and a NOTE OFF MIDI event (i.e. the elapsed time from the moment a note is pressed to the moment of release). The *delta* time is the elapsed time between two NOTE ON events (Fig. 6.2).

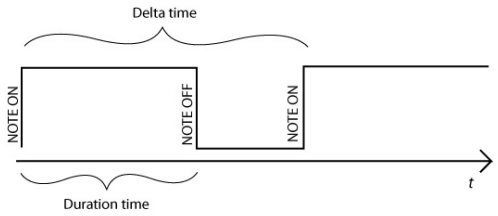


Figure 6.2 MIDI note events scheme. This scheme illustrates a series of two notes. The duration time is relative to the note while the delta time is the elapsed time between two subsequent note attacks. We compute the rest in the middle as *Rest duration = Delta time – Duration time*.

Using this information it is possible to obtain a sequence of notes durations and rests durations that represents a rhythmic phrase. As the duration and delta times follow each other in time the computed notes and rests are immediately quantized. The quantization process provides a minimum note/rest duration step expressed in milliseconds equal to Δ. This value is calculated as follows:

$$\Delta = \frac{60000 / bpm}{24}$$

Where the bpm (beats per minute) is a user parameter. All notes and rests are quantized as a multiple of Δ. Table 6.1 shows some examples. If the user plays notes or rests between multiple values of Δ the note is quantized to the closest beat fraction.

Table 6.1 - Δ encoding example.

Note name	Symbol	Δ Value
Sixteenth note		Δ * 6
Eighth note		Δ * 12
Quarter note		Δ * 24
Half note		Δ * 48

6.3 Rhythmic figure segmentation

When the buffer size of the quantized notes and rests reaches a certain threshold the buffer content is processed in order to identify the states to be used in the Markov

model. The minimum rhythm state length is equal to one beat. This guarantees that a rhythmic continuation will be kept linked to the main rhythmic accents (beats) in the measures. In fact, if one single note duration is used as a state the model based phrases risks to be too erratic. For example consider the following rhythm played by during the learning phase and used by the system to build the rhythm model (the time signature is 4/4):

If we build a statistic model with single notes as states a sequence of prediction based on the model could produce the following rhythm:

This sequence is a combination of the two identified states (the quarter and the sixteenth note) but unlike the original sequence is not linked with the main beats of the measure and introduces syncopation. In fact, the original rhythmic figure presents quarters "on the beat" while in the continuation figure the quarters are shifted by a sixteenth note. The musical effect is completely different. We want to produce syncopations only if the musician played them. Therefore the minimum state length captured is of one beat (a quarter) or a multiple. Fig. 6.2 illustrate an example of phrase segmentation.

Figure 6.3 - Rhythmic figure segmentation. The sequence is segmented in parts equal to a beat or a multiple of it. In this case the beat is equal to a quarter note.

When during segmentation a new segment is identified the system checks whether the segment consists in an already visited state. If the segment is a new state the system associate a letter from the alphabet as a key and inserts state and key in a hash table. For example the first group in Fig 6.2 is quantized and encoded as *[36Δ; 12Δ]*, assigned to the symbol *a* and inserted in the hash table. Then a learning string is updated: *learning*

string = "a". On the other hand if the state has been already visited the relative key is retrieved and once again the learning string is updated.

6.4 Model construction and rhythmic phrase continuation

When the learning phase is over the system uses the learning string to create the model. As in the previous chapter the *VMMPredictor* is used as an implementation of a Variable Length Markov Model. This model is used to make predictions on future states using a variable context length.

The production of new rhythmic figures consists of the prediction of new rhythmic elements. Multiple subsequent predictions provide similar patterns from a statistical point of view. The prediction is based only on *visited states* present in the hash table. In other words predictions of rhythmic elements not present in the learning string and hence with probability equal to zero are not requested. When all the predictions based on a certain context (i.e. the previous rhythms played by the musician or proposed by the system) are available a random number between zero and one is generated and the correspondent symbol is selected. The symbol is decoded and inserted in a buffer. The process described in this paragraph is repeated a number of times in order to fill the output buffer.

This buffer is read in association with the buffer containing melodic patterns. Reading the rhythm buffer gives information about the arrangement in time of the notes in the melodic patterns buffer. In this way all the data needed to produce an improvisation continuation is read by a MIDI sequencer. The sequencer assembles the rhythmic values with the notes' pitches and sends them to the MIDI interface (Fig. 6.4).

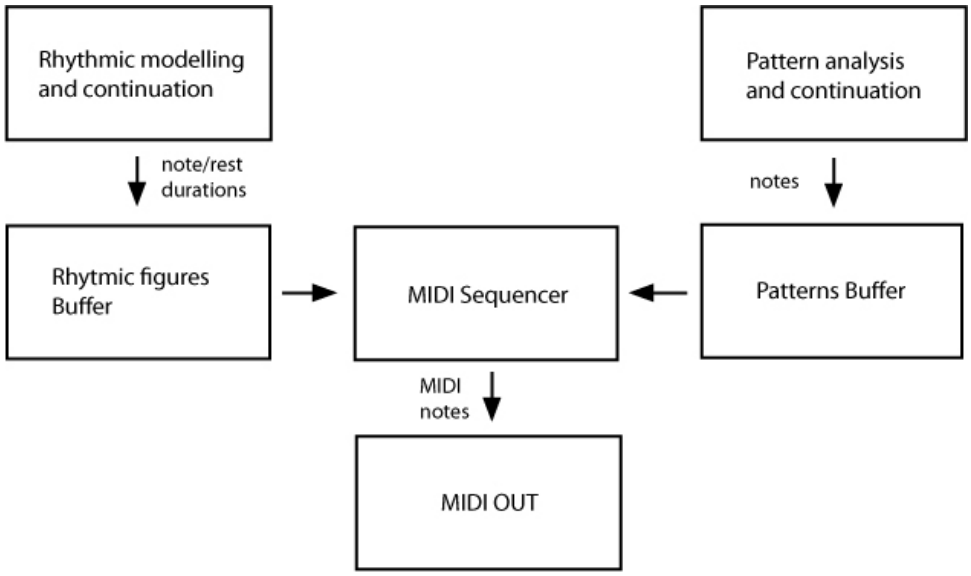


Figure 6.4 - Block diagram illustrating the converging of rhythmic and melodic data. New rhythmic figures and melodic patterns are produced and collected in their respective buffers. Then a MIDI sequencer module reads the buffers and produces a sequence of MIDI notes.

Chapter 7

Implementation and testing

7.1 Development environment

The main environment used for the implementation of the algorithms and methods described in the previous chapters is *Max*. *Max* is a visual programming language for music and multimedia, with numerous features that make it suitable for artists and audio engineers [22]. Among its desirable features are its modularity and the availability of many routines for multimedia signals processing. It has an extensible design and graphical interface where the program structure and the GUI are presented to the user simultaneously.

The way of programming in Max is that of a data-flow system: Max programs are called *patches* and are made by arranging and connecting building blocks of *objects* within a *patcher*, or visual canvas. These objects act as self-contained programs (in reality, they are dynamically-linked libraries), each of which may receive input through one or more visual *inlets*, generate output through visual *outlets*, or both. Objects pass messages from their outlets to the inlets of connected objects. *Max* supports six basic atomic data types that can be transmitted as messages from object to object: *int*, *float*, *list*, *symbol*, *bang*, and *signal* (in the case of *MSP* audio connections). A number of more complex data structures exist within the program for handling numeric arrays (*table data*), hash tables (*coll data*), and XML information (*pattr data*). The order of execution for messages traversing through the graph of objects is defined by the visual organization of the objects in the *patcher* itself. As a result of this organizing principle, Max is unusual in that the program logic and the interface as presented to the user are

typically related, though newer versions of Max provide a number of technologies for more standard GUI design. Max comes with hundreds of *objects* as the standard package and extensions to the program can be written by third-party developers as Max patchers or as objects written in C, C++, Java, or JavaScript.

The implementation of the functionalities described in this work is based on the development of external objects. The *Max* environment makes easier to process MIDI signals and manage MIDI devices, MIDI files and temporal events in general. In fact, the approach based on visual connections between objects is closer to the way of thinking of inputs and outputs of various components. On the other hand classical algorithms that could be written in a few lines of code become intricate from the data-flow point of view. Therefore the programming language Java was used within the development environment Eclipse, a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. The main components (the *metronome module*, the *chord recognizer module*, the *progression module*, the *modal analysis module* and the *pattern module)* are written in Java and then integrated as external objects in the *Max* environment.

7.2 Prototype 1 - Modal analysis

The first implemented prototype is relative to the analysis of *modes* during an improvisation performance on a harmonic progression. This prototype provides a user interface built in the MAX environment:

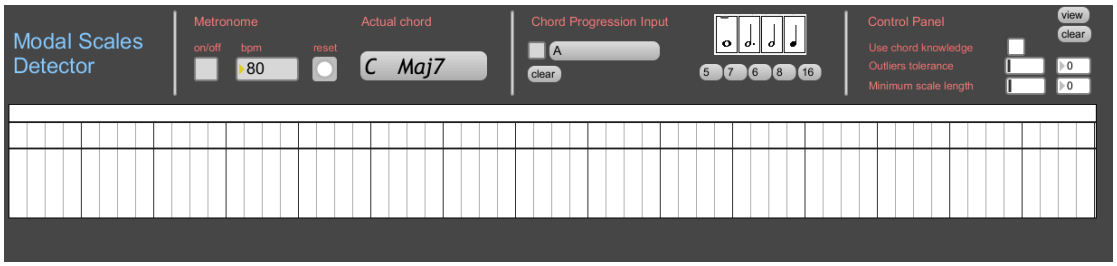


Figure 7.1 - Modal analyser GUI

In the upper part of the GUI there is a control panel. The first parameters are dedicated to the metronome. The user can switch on and off the metronome, set the “bpm” and reset the entire progression in order to restart the session from the beginning. Besides the metronome settings there is a display with the current chord.

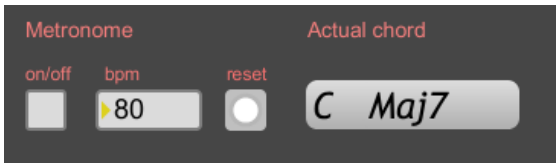


Figure 7.2 - Metronome panel

At the centre of the control panel are the buttons dedicated to the chord recognizer. The chord recognizer allows the insertion of a harmonic progression in the system simply playing the chords on the keyboard and selecting the relative duration. If the recognizer is active the tonic and type of the chord being played is displayed. At this point the user can select the duration (in quarters) from the note icons or, in case of a longer duration, pressing on a number button. The "clear" button allows deleting the current progression and starting with a new insertion.

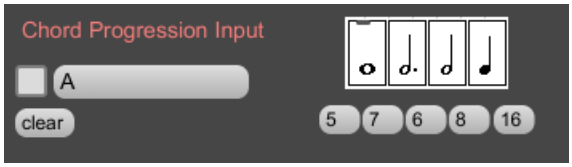


Figure 7.3 - Chord recognizer panel

Finally, on the right side of the panel there are some parameters relative to the segmentation of the sequence of played notes. The first option with label "Use chord knowledge" has the following function: if the option is selected the system performs mode detection using the knowledge of scales compatibility with the current chord. In other words if the user plays "wrong" scales these will not be detected because the algorithm search correspondences with the "right" scales. If the option is left unselected the system uses the entire scale database in order to perform selection. The second parameter is set with a horizontal slider. The user can choose the percentage of allowed outliers in a scale. If the tolerance is 0% the scales have to be played perfectly, without errors or the detection will fail. Another slider allows selecting how many notes are required for the detection of a specific mode. This parameter avoids over-segmentation of the notes played during the session.

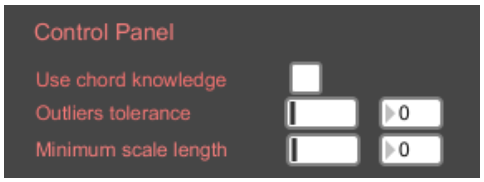


Figure 7.4 - User parameter for segmentation control

On the white chart under the control panel the system displays the results. The black vertical lines represent the division between measures. The gray vertical lines represent the quarter division. During the insertion of the chord progression the system updates the chart displaying the chords name and the suggested scales for that chord. The suggested scales are displayed with different colours in order give an indication on the sound: brighter colours represents a brighter sound while darker colours are associated to scales with a more obscure sound. The following picture illustrates the chart with chord name and the suggested scale. If the "Use chord knowledge" button is selected the user should use the suggested scale otherwise no scales will be detected. If the user wants to improvise and change some modes freely he deselects the option and uses whatever scale.

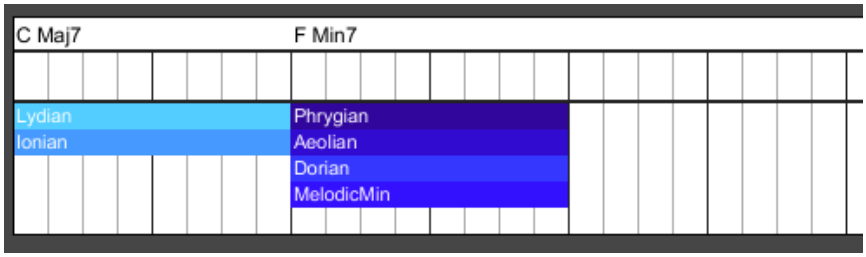


Figure 7.5 - Harmonic progression with chord names and scales suggestions

At the end of each chord duration, the detected modes are displayed in the region between the black horizontal lines. A red line over the chords name indicates the current point in the harmonic progression during session.

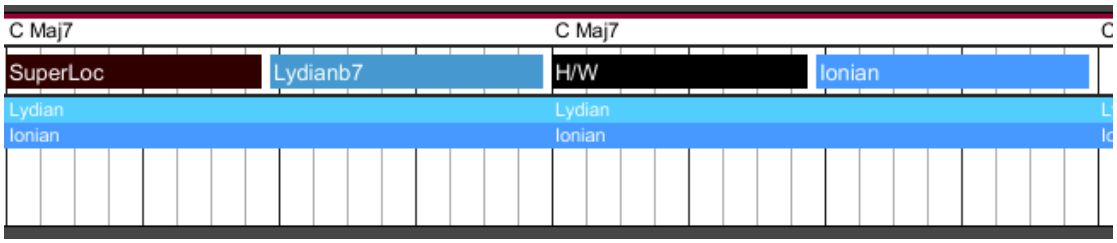


Figure 7.6 - Segmentation of a sequence of notes in four modes displayed with different colours

The functionalities here described are implemented both with *Max* standard objects and Java-written external objects. As illustrated in Fig. 7 both type of object are connected in a *Max Patcher*

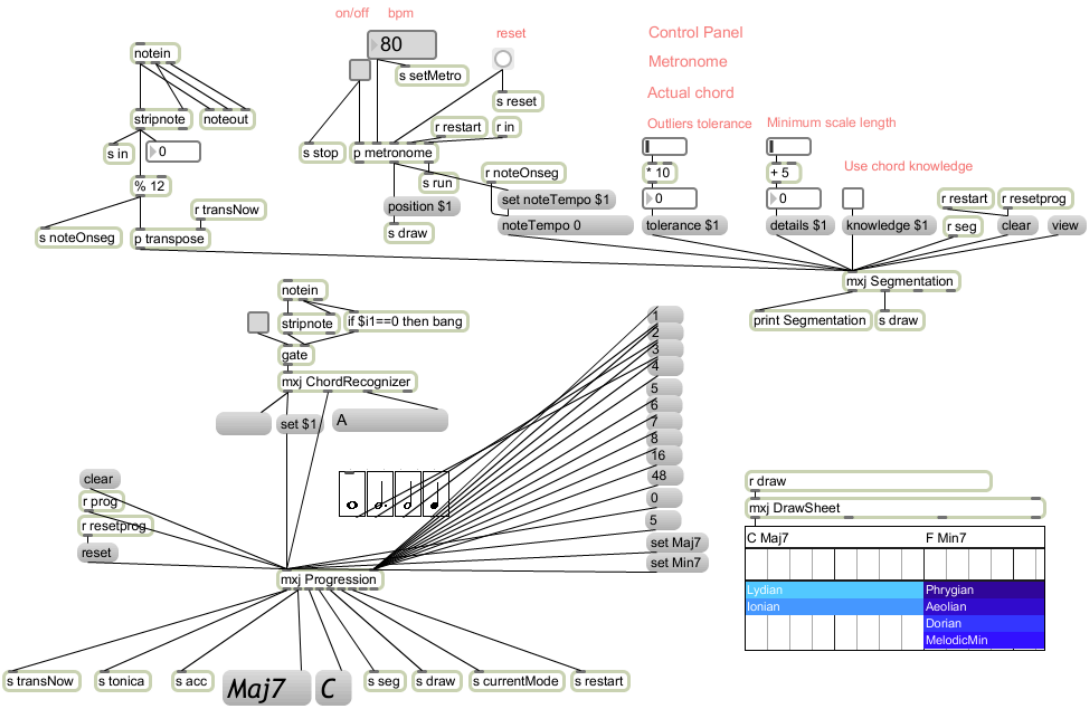


Figure 7.7 - Max patcher for the modal analysis. This figure illustrates a screen shot of the Max environment where the prototype was implemented. The objects with the "mjx" prefix name are external objects implemented in Java (Segmentation, Progression, ChordRecognizer, DrawSheet). These objects are connected to standard Max objects.

7.3 Prototype 2 – Change of the mode-space trajectory

The second prototype is implemented in order to test the ability of the system to project melodic patterns on different modes. The objective is hearing the effect of changing the underlying mode of a melody or improvisational pattern. The prototype is implemented in Max with Java external objects, too. The main engine is based on the module for *modal analysis* and has an additional module that implements the technique based on the *modes independent* representation described in Chapter 5. In this case there is no automatic continuation, so the note sequence is made *mode independent* but it is stored "as it is" without building any statistical model. The system has two operating mode: a **learning mode** and a **running mode**. It has no graphical user interface and can be controlled with the MIDI keyboard. The user can switch between the two operating modes pressing the lowest black key. The MIDI keyboard has a split point (F#3): the

lower part is dedicated to the accompaniment and the upper part is dedicated to the melodic patterns.

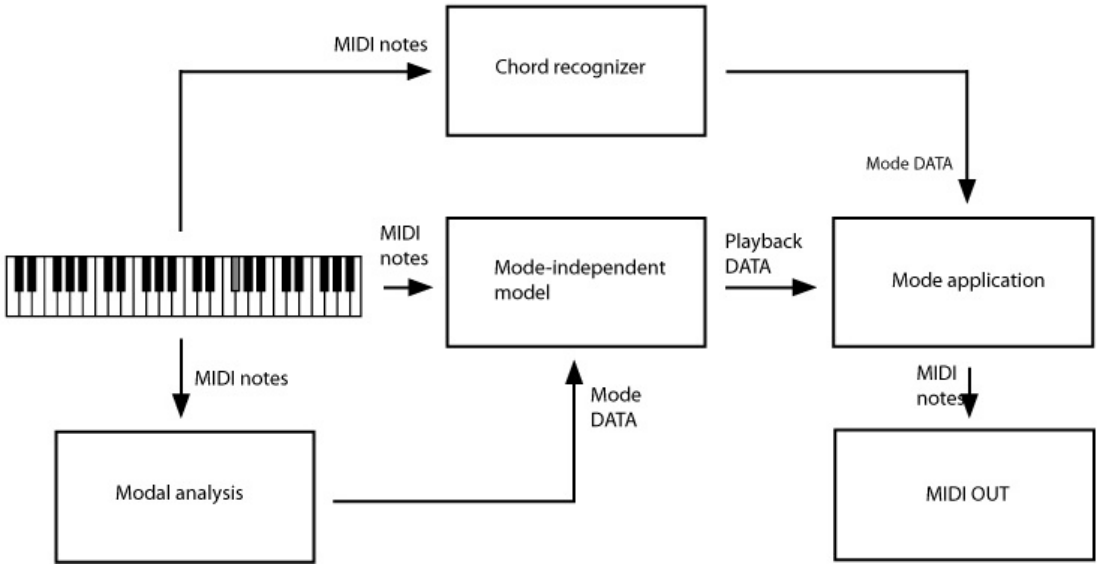


Figure 7.8 - Prototype 2 block diagram. This diagram illustrates the data-flow between the main components of prototype 2. The notes coming from the MIDI interface are used for mode detection (Modal analysis), for pattern/mode orthogonalization (Mode independent model) and for changing the mode during the running phase (Chord recognizer). The Mode application component applies the melody/pattern to the current mode in real-time and sends real notes to the MIDI OUT interface.

A typical session with this prototype can be described in four steps:

1. Initially the system is in **learning mode**. The user selects a mode playing a chord or some sort of accompaniment in the lower part of the keyboard and starts to play the desired line in the upper part.
2. When the user finishes he presses the lowest black key on the keyboard and the system goes in running phase. Through modal analysis the underlying mode is detected and the played line is stored in a mode independent way. Then the system playbacks the line on the same mode.
3. During the running phase the user can interact with the system playing chords in the lower part of the keyboard. These chords are immediately recognized and a compatible mode is selected. Without interrupting the playback session the system project the remaining part of the line on the new mode.
4. Pressing the lowest black key on the keyboard the user sets a new learning phase (back to step 1).

7.4 Prototype 3 – Continuation and change of the mode-space trajectory

The third prototype is implemented in order to verify the effectiveness of a Variable-Length Markov Model based on the *mode independent* representation described in Chapter 3. This model is used to produce a continuation of an improvisation with a coherent style from the pattern point of view in relation with a learning phase. This functionality is associated with the possibility of changing the trajectory of the improvisational lines in the mode space presented in the previous prototype. The development environment is once again Max with external Java-written objects.

The main components are the *Modal analysis* module and the *Pattern and continuation* module. In the latter module the techniques for the mode-independent representation and modelling with VLMMs described in Chapter 5 are implemented. The system is controlled just switching between learning mode and running mode as described for the previous prototype. The big difference of this implementation respect to the previous prototype is that the system does not playback the learned sequence but performs an automatic continuation of the improvisation using the VLMM to predict the future notes in output. Therefore, the various phases are summarized as follows:

1. **Learning mode:** the user improvises with the right hand and performs the desired accompaniment with the left hand using a single mode.
2. The user presses the lowest black key and system goes in **running mode**. Modal analysis is performed and the VLMM is built. The system starts to improvise making prediction based on the model.
3. The user interacts changing the underlying harmonic progression without any constraint relative to chord types or temporal sequence.
4. Pressing the lowest black key on the keyboard the user sets a new learning phase (back to step 1).

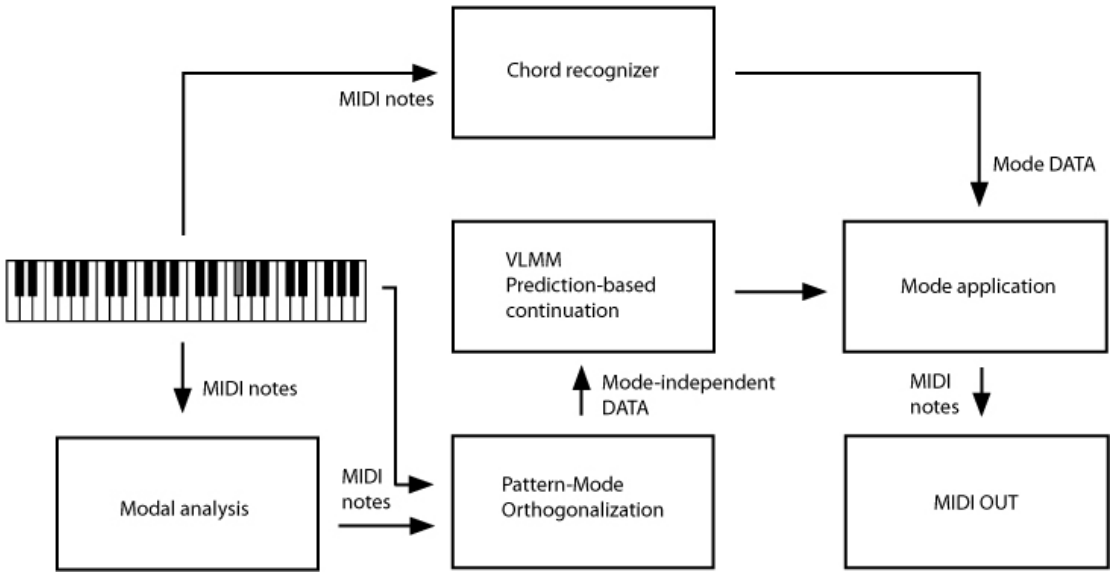


Figure 7.9 - Prototype 3 block diagram. This diagram illustrates the data-flow between the main components of prototype 3. It is similar to the diagram of prototype 2 except for the VLMM and continuation component. In this case the data arriving to the Mode application component is not a simple playback but an automatic improvisation.

7.5 Modal analysis test – Prototype 1

7.5.1 Test n. 1

This test is based on the first 8 bars of the jazz standard "Take the A Train" by Duke Ellington and Billy Strayhorn (Fig 10). The inserted harmonic progression is: Cmaj7 (2 bars), D7 (2 bars), Dm7 (1 bar), G7 (1 bar) and Cmaj7 (2 bars). The test consisted in playing on the progression using an outliers tolerance of 10%. On the longer chord two modes were played. The test was repeated two times changing the "Use chord knowledge" parameter. The results on mode detection are given by the system in the form of a mode name and an extension of that mode in the harmonic progression. The harmonic progression can be viewed as chords disposed on a time axis. Therefore the main measurements made during the test are relative to the percentage of time axis covered by the correct mode (Correct detection %), the percentage of the time axis covered by a wrong mode (Incorrect detection %) and the percentage of the time axis where a mode was played but nothing has been detected by the system.

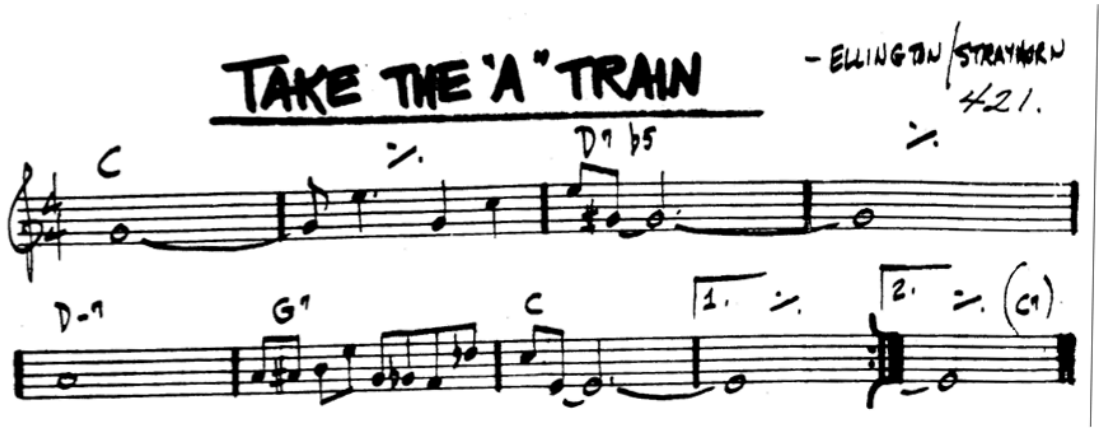


Figure 7.10 - First eight bars of the jazz standard "Take the A Train"

Table 7.1 - Results of test n.1. The test consisted in playing the first 8 bars of "Take the A train". The second column represents the percentage of time where the correct modes have been detected. The third columns represents the percentage of time where a mode has been detected but it wasn't the right one. The last column represents the percentage of time where a mode was played but nothing has been detected by the system.

Use chord knowledge	Correct detection	Incorrect detection	False negative
ON	98%	2%	0%
OFF	97%	3%	0%

The results show that if the user plays correctly the scales (even changing scale on the same chord) the system performances are excellent both in case of a reduced scale possibility (*Use chord knowledge* ON) and considering all the scales (*Use chord knowledge* OFF). When the "*Use chord knowledge"* parameter is on the system use the chord type information to select a restricted set of modes. In any way none of the chords presented in the sixty-chords system (Chapter 1) can indentify only one scale. Therefore the system still has to detect the correct mode. Furthermore, if the parameter is off only the tonic is determined and the system has to discriminate between all the modes in that key. Thus, this result is a positive sign about the system robustness.

7.5.2 Test n.2

This test is based on the massive insertion of outliers in case of chord with a reduced scale possibility (Major chord, Semi-diminished chords etc) and activating the "Use chord knowledge" parameter. The outliers that don't belong to any compatible scale are discarded. In this manner the user can play how many outliers of type A (refer to Chapter 4) as he wish without affecting the detection of the underlying scale.

Table 7.2 - Results of test n.2 This test was performed playing on chords with a reduced scale possibility. The results show that the system correctly ignores the outliers that don't indicate a mode change.

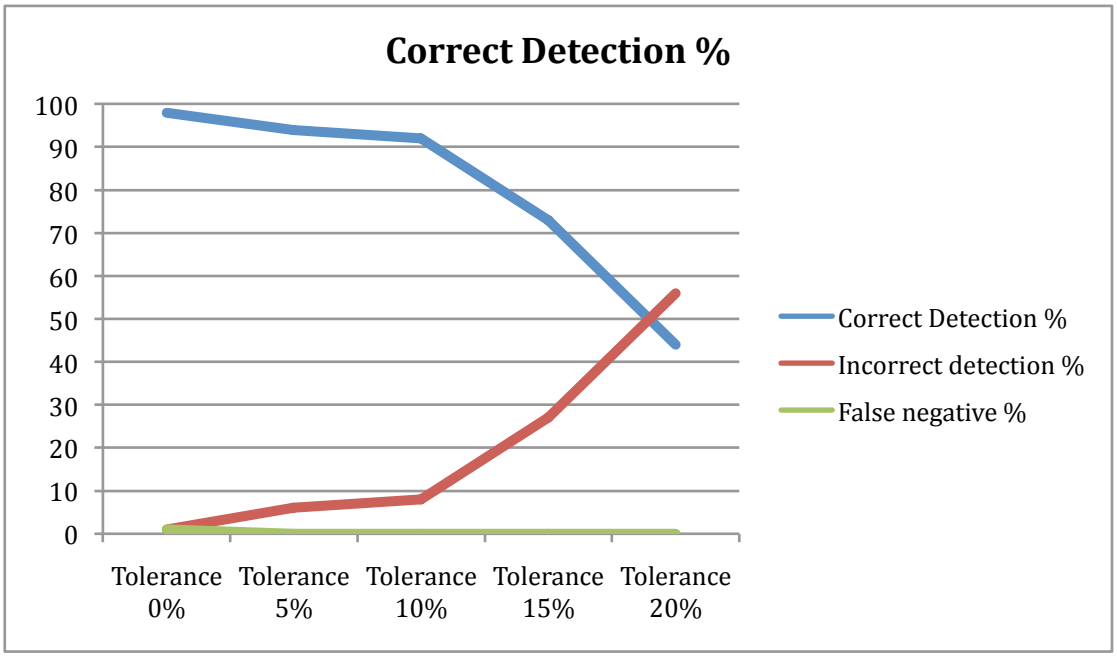
Use chord knowledge	Correct detection	Incorrect detection	False negative
ON	99%	1%	0%

7.5.3 Test n.3

This test is performed in order to verify the robustness of the *Modal analysis* prototype system in case of a total free improvisation were only the fundamental note is determined. In this case the user can play every mode regardless of the chord type or function. The test was performed playing 10 times on an 8 bars C pedal. The average mode duration was 2 bars and the number of played notes for each mode was between 32 and 40. The "Minimum scale length" was set to 14. The test is repeated 5 times changing the "Outliers tolerance" parameter from 0% to 20%.

Outliers tolerance	Correct detection	Incorrect detection	False negative
0%	98%	1%	1%
5%	94%	6%	0%
10%	92%	8%	0%
15%	73%	27%	0%
20%	44%	56%	0%

Figure 7.11 – Results of test n. 3. This test consisted in the segmentation of a total free improvisation were only the fundamental note was determined. Therefore no information on the chord type could help the system in recognizing the right mode.



The results of this test show that the system has excellent performance when the tolerance to outliers is kept low and the user plays mostly "within the mode". In this case the user can choose any scale and often the modes can differ in one or two notes. This requires that the user is coherent with the modes he chooses so that the tolerance parameter can be maintained under the 10%. This way the mode detection will be very precise both in time and in the mode space. Keeping the tolerance between 10% and 15% gives more freedom to the player to include "wrong notes" here and there, such (e.g. chromatic embellishments) but the detection becomes slightly less accurate over time. Mode detection, on the other hand, keeps performing well. This time the reduction of performance is only in the mode transition: when the user changes the scale the system considers the first outliers in the tolerance percentage and so has a little delay (usually 1 or two quarter) in detecting the next mode. Over 15% the system is not reliable anymore neither in time nor in the mode space. This low result is not attributed to the system but is due to nature of modes: often the modes have a difference in terms of notes under the 20%. So other scales can be selected has candidate since they meet the constraints.

7.6 Pattern projection test – Prototype 2

The testing of prototype 2 was based on different application examples. During the testing sessions some audio excerpt were produced. These excerpt were the object of an evaluation questionnaire proposed to an audience of 20 persons evenly distributed

among musicians and sound engineers. Since the questions are based on musical concepts these were accurately explained to the audience.

7.6.1 Test n.1

The first test consisted in playing the first fragment of the well-known theme from *Bolero* by Maurice Ravel. The melody was played in the upper part of the keyboard with a simple piano sound while the accompaniment consisted in a C major chord played in the lower part with a *string* sound. Then the system repeated many times the melody applying the mode-independent representation every time to a different mode. The chords played by the author in the lower part of the keyboard during the playback guided this change of mode. This example served as training for the audience on the concepts relative to the *modes*. The relative question wanted to investigate the perceived potential didactic value of such a tool:

Do you think that altering the underlying scale (mode) of a recorded melody could be an interesting editing or composing tool?

Not at all	Of limited interest	Moderately	Yes	Definitely yes
0%	0%	18.5%	63%	18.5%

Do you think that altering the underlying scale (mode) of a recorded melody could be an interesting didactic tool?

Not at all	Of limited interest	Moderately	Yes	Definitely yes
0%	0%	18%	45%	36%

7.6.2 Test n.2

The second test consisted in a jazz pattern played by the author on a major scale. The system repeats the pattern on different modes guided by the chords played by the author in the lower part of the keyboard during the playback. This excerpt was created in order to investigate the level of adaptation of the same pattern on different scales.

How do you rate the level of adaptation of the same pattern to different modes?

Null	Poor	Sufficient	Good	Excellent
0%	0%	0%	37%	63%

7.7 "Continuation" tests – Prototype 3

Similarly to the previous test different audio excerpt for protype 3 were produced. These excerpt were the object of an evaluation questionnaire proposed to an audience of 20 persons evenly distributed among musicians and sound engineers. The musical concepts contained in the questions were accurately explained to the audience.

7.7.1 Test n.1

This test was based on learning the improvisational style from a MIDI file of *Peace Piece* by Bill Evans. This is a piano solo composition mainly based on the Ionian mode. After one minute of learning phase the system proposes a continuation while the author plays the same accompaniment executed by Bill Evans in the original piece.

Do you think that there is continuity between the first part and the automatic continuation?

Null	Poor	Sufficient	Good	Excellent
0%	0%	0%	50%	50%

How would you rate the style coherence of the automatic continuation in relation with the original part?

Null	Poor	Sufficient	Good	Excellent
0%	0%	10%	80%	10%

How would you rate the level of continuation adherence to the underlying harmonic progression?

Null	Poor	Sufficient	Good	Excellent
0%	0%	10%	30%	60%

7.7.2 Test n.2

This test was based on a short learning phase consisting in different patterns played in the author personal style. Furthermore the user interacts with many chord changes during the running phase using a *pad* sound (a sound with a smooth timbre used to determine the harmony without interfering with the leading sound) (a continuous background timbre for harmonic progressions) in the lower part of the keyboard. This session has many analogies with a duet where the improviser doesn't know in advance

the chord progression on which he's going to improvise. Hence he adapts his solo recognizing *by ear* the new chords and adapting the assolo.

Do you think that there is continuity between the first part and the automatic continuation?

Null	Poor	Sufficient	Good	Excellent
0%	0%	20%	40%	40%

How would you rate the style coherence of the automatic continuation in relation with the original part?

Null	Poor	Sufficient	Good	Excellent
0%	0%	10%	30%	60%

How would you rate the level of continuation adherence to the underlying harmonic progression?

Null	Poor	Sufficient	Good	Excellent
0%	0%	0%	30%	70%

7.7.3 Test n.3

The last test is based on another piano improvisation with a completely different style from the scale point of view. In fact, here the pentatonic scale is the main resource and chords built by fourths characterize the accompaniment. The resulting sound is well defined. The harmonic progression remains static also during continuation so the main objective is to investigate the style coherence of the automatic improvisation.

Do you think that there is continuity between the first part and the automatic continuation?

Null	Poor	Sufficient	Good	Excellent
0%	0%	10%	30%	60%

How would you rate the style coherence of the automatic continuation in relation with the original part?

Null	Poor	Sufficient	Good	Excellent
0%	0%	0%	30%	70%

How would you rate the level of continuation adherence to the underlying harmonic progression?

Null	Poor	Sufficient	Good	Excellent
0%	0%	0%	20%	80%

The answers to the questionnaire show that the proposed applications of the modal analysis and pattern/mode orthogonalization are seen as interesting tools for the didactic, the composing and the editing field. This result encourages continuing with the researches in this direction. The tests on the improvisation continuation gave definitely positive results, too. The musical knowledge base can make the difference in continuing the improvisation on a specific style (see the "Pentatonic" improvisation example). However, the process of style modelling from the patterns point of view needs additional work. Very good results, lastly, for the *adherence to the underlying harmonic progression*. In fact, the system responds very quickly to the chord changes played by the user and the knowledge base guarantees high coherence between chords and modes.

7.8 A preliminary assessment of rhythmic analysis and pattern generation

Of the rhythmic analyser we made only an early assessment in order to verify that the structural choices would prevent the system from accumulating undesired rhythmic errors. The application of a rhythmic modelling system to the prototypes previously described allowed an extension of the automatic improvisation also to the rhythmic dimension. Therefore, instead of performing a mere playback of figures from the training sequence the system was capable to produce new rhythmic phrases. The technique based on the selection of states for the Variable Length Markov model with a minimum length of a beat (in our case a quarter note) gave good results. In fact, the misalignment of the phrases in relation to the beats and the presence of syncopation not introduced in the training sequence were acceptable. Furthermore the segmentation described in Chapter 5 allowed capturing complex rhythmic divisions such as, for example, half note triplets. Unfortunately, the lack of expressiveness due to the absence of MIDI velocities modelling and due to quantization makes the rhythmic continuation not yet satisfactory. Often the human rhythmic interpretation is very complex and based on nuances. The variables are numerous and even two slightly different rhythmic phrases can arouse a completely different effect. An improviser can play very precisely on the beats or play *rubato*. Furthermore he can obtain a total different effect moving

just a couple of accents in the phrase. Therefore the dimensions of expressiveness, accentuation and time interpretation need further work in order to create a complete and effective model.

Chapter 8

Conclusions and future work

The main goal of this work was to understand which way to go for modelling complex processes such as those that govern jazz improvisation. The first issue that we addressed issue was to understand how improvisation actually works, and build a model that complies with this understanding. We found that the improvisational process is not free, but relies on a rich model-based structure. The musician, in fact, progressively builds this structure layer by layer by incorporating rules and constraints, and turning them into generational models through constant "drilling" (practicing patterns that obey such rules). The availability of rule-based generational models frees the musician from the burden of having to think in terms of "what notes to play", and allows him/her to work on a more abstract level, where what matters is the management and the interplay between generational/structural rules and choice of patterns to perform.

The fact that the improvisational process is the result of the application of generational rules is well-understood by jazz musicians, who spend a great deal of time practicing on rule-based patterns in order to become proficient in their art. Nonetheless, little is available in the literature which attempts to "model" jazz improvisation according to this principle, i.e. in a rule-based fashion. Therefore the aim of this thesis was to explore the approach of analysing and modelling a jazz improvisation performance with a rule-based generational model. This work takes the first step in this direction by building a *mode-based* generational model. This model relies on a

knowledge base of *modes* used by jazz musicians, a modal analysis tool and a statistical model based on a pattern/mode orthogonalization. The modal analysis, made possible by the knowledge base, provides a high level of interpretation of a sequence of improvised notes. This interpretation, strictly connected with the musician's generational process, is used in a pattern/mode orthogonalization process: the played lines (melodies, patterns etc.) are encoded with a mode-independent representation. Then a statistical model based on a Variable Length Markov Model allowed performing pattern analysis and continuation through predictions. Once again the knowledge base turned out to be fundamental in applying the learned patterns to new harmonic progressions.

The implemented stand-alone modal analysis tool demonstrated an excellent performance during the tests and confirmed to be a good didactic tool. The system has great flexibility since it allows the insertion of the desired harmonic progression and set of modes. Therefore it can be used as "assistant" during the practice session of the aspiring jazz musician. Furthermore the other two prototypes showed good performance in emulating and continuing a jazz improvisation. The audio excerpts prepared for the tests had a remarkable perceptive impact on the audience. However, at the same time, these tests pointed out the main limits of the system. The element that determines the gap between human and machine performance is the rhythmic dimension. In fact, the rhythmic model does not capture the rhythmic subtleties and nuances played by a real jazz musician. Another important limit refers to expressiveness: this first approach to jazz improvisation continuation didn't include a model for that dimension. In conclusion, the obtained results are encouraging and show that the idea of modelling and implementing jazz improvisation as the application of generational rules is a direction that is worth to be further explored.

Future works could expand this system in many directions. First of all the rhythmic model needs further improvements in order to propose more realistic and beat-linked rhythmic figures. The constraint on the time signature (4/4) could be released in order to let the user insert the desired time signature (3/4, 5/4, 6/8 and so on). A dedicated model is requested to manage the expressiveness of the performance that, in the case of a MIDI interface, consists in analysing and modelling the MIDI velocities. In this case the objective is to understand where and how the musician places musical accents in his phrases. Then the knowledge base can be progressively enriched with more rules and constraints, for example with concepts related to the most important notes inside a mode, or techniques for out-of-the-harmony patterns

construction. Finally, the system interface can be expanded in order to accept not only MIDI but also audio inputs extending the use to a broader class of musicians.

Appendix A

Markov Models

The statistical methods of Markov models [3] were initially introduced and studied in the late 1960s and early 1970s. These methods have become increasingly popular in the last several years for many reasons. This success lies basically on the rich mathematical structure and the effectiveness in practical application. These properties make the Markov models the right tool for a wide range of application. They are applied in a number of ways to many different fields. Often they are used as a mathematical model from some random physical process; if the parameters of the model are known, quantitative predictions can be made. In other cases, they are used to model a more abstract process, and are the theoretical underpinning of an algorithm.

A.1 Markov Chains

A Markov chain is a discrete-time random process with the Markov property. Usually, the term "Markov chain" is used to mean a Markov process with a discrete state-space. A discrete-time random process means a system which is in a certain state at each step, with the state changing randomly between steps. The steps are often thought of as time, but they can equally well refer to physical distance or any other discrete measurement. The Markov property states that the conditional probability distribution for the system at the next step given its current state depends only on the current state of the system, and not additionally on the state of the system at previous steps. The changes of state of the system are called transitions, and the probabilities

associated with various state-changes are called transition probabilities. The set of all states and transition probabilities completely characterizes a Markov chain.

Consider a system which may be described at any time as being in one of a set of N distinct states, S_1, S_2, S_3, …, S_N. The next figure illustrates the relative graph.

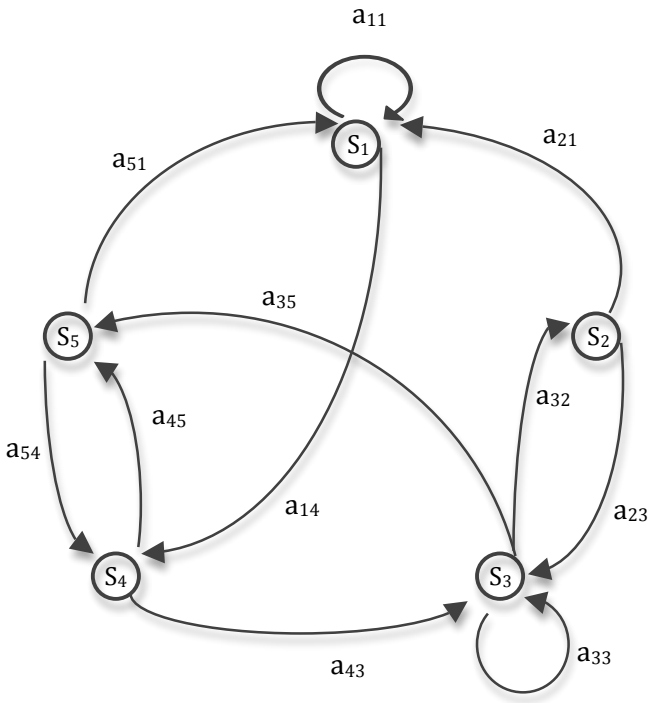


Figure 8.1 – An example of graph for a Markov chain of order 1. S_n represent the states, a_{ij} are the transition probabilities.

At regularly spaced discrete times, the system undergoes a change of state (possibly back to the same state) according to a set of probabilities associated with the state. The time instants associated with state changes are denoted as t = 1, 2, . . . , and the actual state at time t is denoted as q_t. A full probabilistic description of the above system would, in general, require specification of the current state (at time t), as well as all the predecessor states. For the special case of a discrete, first order, Markov chain, this probabilistic description is truncated to just the current and the predecessor state, i.e.,

$$P[q_t = S_j \mid q_{t-1} = S_i, q_{t-2} = S_k, \ldots] = P[q_t = S_j \mid q_{t-1} = S_i]$$

Furthermore we only consider those processes in which the right-hand side of the formula above is independent of time, thereby leading to the set of state transition probabilities a_{ij}, of the form

$$a_{ij} = P[q_t = S_j \mid q_{t-1} = S_i]$$
$$1 \le i, j \le N$$

with the state transition coefficients having the properties

$$a_{ij} \ge 0$$
$$\sum_{j=1}^{N} a_{ij} = 1$$

The above stochastic process could be called an observable Markov model since the output of the process is the set of states at each instant of time, where each state corresponds to a physical (observable) event.

One way of estimating the transition probabilities from the data is to use the observed frequencies of the transitions. This method is illustrated in the following example for a first order Markov chain with state space consisting of the letters of the English alphabet. The process of estimating the transition probabilities involves counting transitions to evaluate: x_i, total number of transitions originating at state S_i; y_{ii}, total number of transitions starting going from S_i to S_i; y_{ik} total number of transitions going from S_i to S_k. The probability of a transition from the state S_1 to S_2 is equal to y_{12}/x_1, the probability of a transition from S_1 to S_1 is y_{11}/x_1.

The sequence

A B C A B B C B

Results in the following estimated transition probability matrix:

	A	B	C
A	1	0	0
B	0	1/3	2/3
C	1/2	1/2	0

The properties described above are respected: each matrix entry is between 0 and 1 and each row has sum equal to 1. With a similar method it is possible to estimate the probability matrix for a second order Markov chain:

	A	B	C
(A,B)	0	1/2	1/2
(B, C)	1/2	1/2	0
(C, A)	0	1	0
(B, B)	0	0	1

Using Markov chains it is generally impossible to predict the exact state of the system in the future. However, the statistical properties of the system's future can be predicted. In many applications it is these statistical properties that are important. In fact, once the Markov chain has been fitted, new state transitions can be simulated using a random number generator. These state transitions will follow the distribution given by the transitions matrix and will therefore be conforming to the original data. The generation of the transition probability matrix, and hence its simulation, is based on local patterns in the data. The simulation will replicate those local patterns in the original data in a manner dependent on the chain order. A higher order chain would capture large sized local patterns in the data and hence produce a simulation that is globally similar to the original data. Conversely, a lower order Markov chain would capture smaller sized local patterns and produce a simulation that is globally less similar to the data.

Markov chains are often employed in algorithmic music composition, particularly in software programs such as *CSound* or *Max*. In a first-order chain, the states of the system become note or pitch values, and a probability vector for each note is constructed, completing a transition probability matrix. An algorithm is constructed to produce and output note values based on the transition matrix weightings, which could be MIDI note values, frequency (Hz), or any other desirable metric.

A.2 Variable Length Markov Models

Improvisation lines often contain complex structures based on patterns. These patterns have a periodic structure with variable length and are connected to each other in many ways. Depending on the note sequence scale different dynamics can be observed. On short scales the local trend is captured while on larger scale more extensive structures are pointed out. Markov models are a natural candidate for music

modelling [5][10] and temporal pattern recognition, mostly due to their mathematical simplicity. However, fixed memory Markov models cannot capture effectively the complexity of such structure due to the pattern variable periodicity. The desired solution is a variable length Markov model (VLMM)[4][8] where the memory length is not fixed and can vary depending on the needs.

The VLMM is an important model that extends the Markov Chain model described in the previous paragraph. This extension basically consists in the variable number of conditioning variables. In a Markov Chain model each random variable in a sequence with a Markov property depends on a fixed number of random variables. On the contrary in the VLM model this number may vary depending on the specific observed realization usually called *context.* The variable length of the context makes this model the right choice for a large class of applications. The VLMM can be defined as follows. Let $\sum$ be a finite alphabet of size $|\sum|$. Consider a sequence with the Markov property

$$x_1^n = x_1 x_2 \ldots x_n$$

of n realizations of random variables, where $x_i \in \Sigma$ is the state at position i $1 \leq i \leq n$ and the concatenation of states x_i and x_{i+1} is denoted by $x_i\, x_{i+1}$. The sequence x_1^n is used as a training set for the VLMM construction algorithm that learns a model P. This model provides a probability assignment for each state in the sequence given its past states. A conditional probability distribution $P(x_i \mid s)$ is generated for a symbol $x_i \in \Sigma$ given a context $s \in \Sigma^*$ where $\sum^*$ represents a sequence of state of length $s \geq 0$. In contrast to N-gram Markov models, which attempt to estimate conditional distributions of the form $\hat{P}(\sigma \mid s)$, with $s \in \sum^N$ and $\sigma \in \sum$, VLMC algorithms learn such conditional distributions where context lengths $|s|$ vary in response to the available statistics in the training sequence.

A.3 Probabilistic Suffix Trees

The application described in this work is one of the countless applications involving machine learning of sequential data, pattern recognition and generation of new sequences through prediction. The class of algorithms for prediction of discrete sequences using VLM models is very large. As described by Begleiter, El-Yaniv and Yona [2] the main algorithms are Context Tree Weighting (CTW), Prediction by Partial Match

(PPM) and Probabilistic Suffix Trees (PSTs). Such algorithms attempt to learn probabilistic finite state automata, which can model sequential data of considerable complexity. Most algorithms for learning VLM include three components: counting, smoothing (probabilities of unobserved events) and variable-length modeling. Specifically, all such algorithms base their probability estimates on counts of the number of occurrences of symbols σ appearing after contexts s in the training sequence. These counts provide the basis for generating the predictor $\hat{P}$. The smoothing component defines how to handle unobserved events (with zero value counts). At last, the variable-length modelling varies depending on the algorithm type. The probabilistic suffix tree (PST) algorithm is well known in the machine learning community. It was successfully used in a variety of applications and is hence used for this application.

The PST prediction algorithm [1] attempts to construct the single "best" D-bounded VMM according to the training sequence. A PST, over an alphabet ∑, is a non empty rooted tree of degree |∑|. This implies a variable degree between zero and |∑| for each node. For each edge in the tree there is a label with a unique symbol from the alphabet. These labels define a unique sequence *s* for each path from a node *v* to the root. The sequence *s* is used in conjunction with $\gamma_s : \Sigma \to [0,1]$ representing the next symbol probability function related to *s*. The pairs (s,γ_s) form the label of the nodes of the tree. For every string *s* labelling a node in the tree the following condition is required:

$$\sum_{\sigma \in \Sigma} \gamma_s(\sigma) = 1$$

The probability that the tree *T* generates a string $r=r_1r_2..r_N$ in $\sum^N$ is

$$P_T^N(r) = \prod\nolimits_{i=1}^{N} \gamma_{s^{i-1}}(r_i)$$

where $s^0=e$, and for $1 \le j \le N-1$, s^j is the string labelling the deepest node reached by taking the walk corresponding to $r_ir_{i-1}...r_1$ starting at the root of *T*. Consider the example brought from [1]. Using the following PST

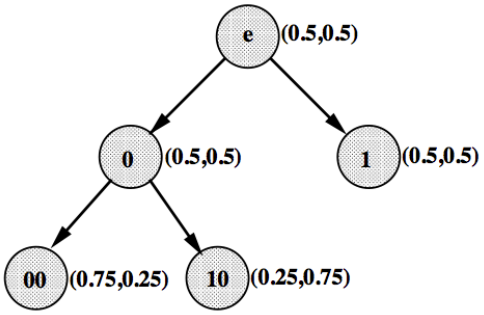


Figure 8.2 - The prediction probabilities of the symbols '0' and '1', respectively, are indicated in parenthesis besides the nodes.

the probability of the string 00101, is 0.5 x 0.5 x 0.25 x 0.5 x 0.75, and the labels of the nodes that are used for the prediction are s^0 = e, s^1 = 0, s^2 = 00, s^3 = 1, s^4 = 10, in view of this definition, by allowing the omission of nodes labelled by substrings which are generated by the tree with probability 0.

The PST learning algorithm has as objective the identification of a good suffix set S for a PST tree and determine a probability distribution $\hat{P}(\sigma \mid s)$ over the alphabet $\sum$ for each sequence $s \in S$. The learning phase of the PST algorithm consists of a process in three steps.

1. A set of candidate contexts is extracted from the training sequence q_1^n . This set forms the candidate *suffix set* $\hat{S}$. Each contiguous sub-sequence *s* of q_1^n of length $l \leq D$, such that the frequency of s in q_1^n is larger than a user threshold will be in $\hat{S}$. This construction guarantees that $\hat{S}$ can be accommodated in a PST tree. The algorithm associates the conditional estimated probability $\hat{P}(\sigma \mid s)$ to each candidate *s*. This association is based on a direct maximum likelihood estimate.

2. The second step consists in the examination of each *s* in the candidates set $\hat{S}$ using a two-condition test. If the context *s* passes the test, *s* and all its suffixes are include in the final PST tree. The test's conditions are:

a) The probability $\hat{P}(\sigma \mid s)$ must be larger than a user threshold in order to guarantee a meaningful connection between the context and the symbol σ.

b) The prediction of σ is made more reliable by the longer context *s* in relation with its *parent* context; If $s = \sigma_k \sigma_{k-1} \ldots \sigma_1$ then its parent context is its longest suffix $s' = \sigma_{k-1} \sigma_{k-2} \ldots \sigma_1$. Considering a user threshold *r* the following condition is required:

$$\frac{\hat{P}(\sigma \mid s)}{\hat{P}(\sigma \mid s')} > r \vee \frac{\hat{P}(\sigma \mid s)}{\hat{P}(\sigma \mid s')} < \frac{1}{r}$$

These conditions must hold simultaneously. If a context *s* passes the test all its suffixes are added to the tree as well and hence don't require further examinations.

3. Finally the probability distributions relative to the contexts are smoothed. An other user-defined constant is required to set a minimum probability if $\hat{P}(\sigma \mid s)$=0.

The PST learning algorithm has different parameters that should be set by the user depending of the application at hand.

The empirical probability function $\hat{P}$ based on a training sequence can be formally defined as follows. For a given sequence *s*, $\hat{P}(s)$ is roughly the relative number of times σ appears after *s*. If we consider a training sequence *r* with length *m,* then for any sequence *s* of length at most *L*, define $\chi_j(s)$ to be 1 if $r_{j-|s|+1}...r_j=s$ and 0 otherwise. The probability for *s* is defined as

$$\hat{P}(s) = \frac{1}{m-L}\sum_{j=L}^{m-1}\chi_j(s)$$

while the probability for any symbol σ is:

$$\hat{P}(\sigma \mid s) = \frac{\sum_{j=L}^{m-1}\chi_{j+1}(s\sigma)}{\sum_{j=L}^{m-1}\chi_j(s)}$$

If there is a training set formed by *m'* sequence $r^1,...,r^{m'}$, each of length $l \geq L+1$ then for any sequence *s* of length at most L, define $\chi_j^i(s)$ to be 1 if $r^i_{j-|s|+1}...r^i_j=s$, and 0 otherwise. The probability for *s* becomes:

$$\hat{P}(s) = \frac{1}{m'(l-L)}\sum_{i=1}^{m'}\sum_{j=L}^{l-1}\chi_j^i(s)$$

and for any symbol σ:

$$\hat{P}(\sigma \mid s) = \frac{\sum_{i=1}^{m'} \sum_{j=L}^{l-1} \chi_{j+1}(s\sigma)}{\sum_{i=1}^{m'} \sum_{j=L}^{l-1} \chi_{j+1}(s)}$$

For simplicity all the training sequences are assumed to have the same length and that this length is polynomial in m, L and $\sum$. The case in which the sample strings are of different lengths can be treated similarly, and if the strings are too long then we can ignore parts of them.

The PAC model for learning Boolean concepts from labelled examples motivates the learning model here described. This ensures that with high probability (≥1−δ), the Kullback-Leibler divergence[1] between the PST estimated probability distribution and the true underlying, but unknown, VMM distribution is not larger than ε after observing a training sequence whose length is polynomial in 1/ε, 1/δ, D, |Σ| and the number of states in the unknown VMM model.

An example based on the training sequence x_1^{11} = **abracadabra** is reported from [2]. In this case the user parameters are P_{min} = 0.001, γ_{min} = 0.001, α = 0.01, r = 1.05, D = 12. Each node is labelled with a sequence s and the distribution ($\hat{P}_s$(a), $\hat{P}_s$(b), $\hat{P}_s$(c), $\hat{P}_s$(d), $\hat{P}_s$(r)). The PST learning phase has a time complexity of $O(Dn^2)$ and a space complexity of $O(Dn)$. The time complexity for calculating $\hat{P}(\sigma|s)$ is $O(D)$.

[1] The Kullback-Leibler distance (KL-distance) [23] is a non-symmetric measure of the difference between two probability distributions P and Q. KL measures the expected number of extra bits required to code samples from P when using a code based on Q, rather than using a code based on P. Typically P represents the "true" distribution of data, observations, or a precisely calculated theoretical distribution. The measure Q typically represents a theory, model, description, or approximation of P.

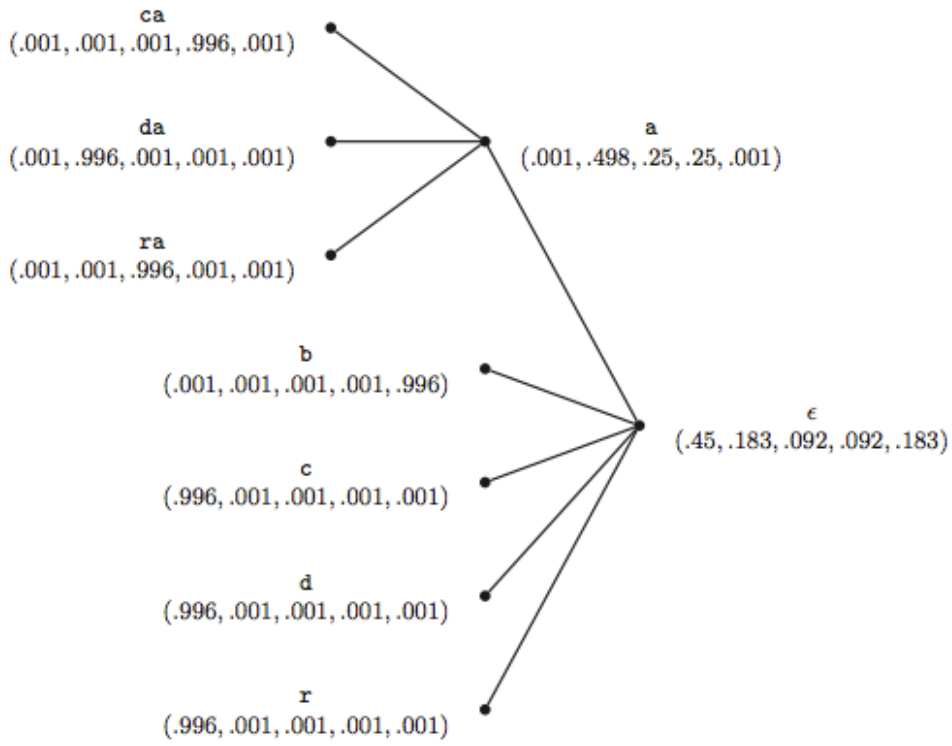


Figure 8.3 - A PST based on the training sequence x_1^{11} = abracadabra

Bibliography

[1] D. Ron, Y. Singer, N. Tishby, *The power of amnesia: Learning probabilistic automata with variable memory length*. Machine Learning, Vol. 25, 1996.

[2] R. Begleiter, R. El-Yaniv, G. Yona, *On Prediction Using Variable Order Markov Models*. Journal of Artificial Intelligence Research, Vol. 22, pp. 385-421, 2004.

[3] L. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, Proc. IEEE 77, 1989.

[4] A. Galata, N. Johnson, D. Hogg, *Learning Variable-Length Markov Models of Behavior*, Computer Vision and Image Understanding Vol. 81, pp. 398–413, 2001.

[5] K. Höthker, *Modelling the Motivic Process of Melodies with Markov Chains*, ICMC Proceedings 1999.

[6] J. Feulner, and D. Hörnel. *Melonet: Neural Networks that Learn Harmony-Based Melodic Variations,* ICMC Proceedings 1994, ICMA Århus 1994.

[7] R. Dannenberg, B. Thom, D. Watson, *A Machine Learning Approach to Musical Style Recognition*. ICMC Proceedings, pp. 344-347, Thessaloniki, 1997

[8] Peter Bühlman, Abraham J. Wyner. *Variable Length Markov Chain.* The Annals of Statistics, Vol. 27, No. 2, pp. 480-513, 1999.

[9] P. Rolland, J. Ganascia, *Automated Motive-Oriented Analysis of Musical Corpuses: a Jazz Case Study*, ICMC Proceedings Hong-Kong, 1996.

[10] F. Pachet, *Playing with Virtual Musicians: the Continuator in practice*, IEEE Multimedia 9(3,) pp. 77-82, 2002.

[11] D. M. Franz. *Markov Chains as Tools for Jazz Improvisation Analysis*. Thesis Master of Science, Virginia Polytechnic Institute and State University, 1998.

[12] B. Thom*. Learning Models for Interactive Melodic Improvisation*. Proceedings ICMC 99. pp. 190–193, 1999.

[13] T. La Fata. *Writing Max Externals in Java.* Manual*,* © 2000-2004 Cycling '74*.* http://cycling74.com/

[14] Y. Marom*. Improvising Jazz with Markov Chains*. Technical report, Department of Computer Science, The University of Western Australia, 1997.

[15] R. Nikolaidis, G. Weinberg, *Playing with the Masters: A Model for Improvisatory Musical Interaction between Robots and Humans.* 19th IEEE International Symposium on Robot and Human Interactive Communication, Viareggio, pp. 712 - 717, 2010.

[16] Luigi Rossi. Teoria Musicale. Edizioni Carrara, 1991.

[17] D. Haerle. *Jazz Improvisation for Keyboard Players.* Studio P/R – Warner Bros. Publications Inc., 1978.

[18] D. Haerle. *Scales for Jazz Improvisation*. Alfred Publishing Co., Inc. 1983.

[19] J. Aebersold. *How to play jazz and improvise.* Volume 1, revised sixth edition, Published by Jamey Aebersold, 1992.

[20] J. Mehegan. Tonal and Rhythmic Principles. Volume I, revised edition. AMSCO Music Publishing Company (1984).

[21] J. Cooker, J. Casale, G. Campbell, J. Greene. *Patterns For Jazz*. Studio P/R, 1970.

[22] Wikipedia. *Max (Software) Wiki Page*:
http://en.wikipedia.org/wiki/Max_(software)

[23] S. Kullback, R.A. Leibler, *On Information and Sufficiency*. Annals of Mathematical Statistics Vol. 22 (1): pp. 79–86, 1951.