Internship report

Computational analysis of jazz chord sequences

Romain VERSAEVEL, M1 Informatique Fondamentale, ENS de Lyon

Tutored by David MEREDITH, Associate professor at Aalborg University,

leader of the Music Informatics and Cognition group

August 27, 2015

Abstract

I report here the results of my work at Aalborg University during the summer 2015. My main task was to provide an analysis of a dataset of chord sequences of 30 jazz songs. The ultimate goal is to understand jazz better thanks to this kind of investigations.

The analysis I propose studies the compression of the dataset, with two algorithms: the LZ77 compression algorithm and an algorithm computing *diagonal patterns*. Their performances in terms of *compression factors* were improved by introducing compression with loss through the use of *similarity measures* between chords. The obtained analysis is successful; it validates my methods and shows room for further research; its main suggests that the structure of jazz chord sequences structure should rather be seen as a whole than linearly.

The algorithms, their results, as well as a short introduction to computer music can be found in the report.

Contents

1 Introduction

This report presents the three-month internship I did as part of my Master 1 of Computer Sciences at ENS Lyon. This internship took place at Aalborg University, Denmark, from May 25th, 2015 to August 14th, in the Department of Architecture, Design and Media Technology. I was supervised by Prof. David Meredith, who specializes in computer music.

During this internship I mainly worked on computational music analysis, and more specifically on the analysis of a dataset of chord sequences from popular jazz songs. And extract of this dataset (presented in section 3.1) gathered by the Sony CSL research laboratory was provided to Aalborg University, in order to be studied.

The report has three main sections. The first one is an introduction to the research field, *computer music*. The second section describes my work on the jazz chords dataset, consisting in two different ways of compressing it, and the obtained results. Finally, the third section presents other related developements I started investigating, together with possible research directions.

An appendix with more details (additional definitions, examples of the algorithms execution, exhaustive numerical results...) can be found on the GitHub repository I used.

2 Computer music

This section briefly introduces the area of computer music. The non-expert reader can find here a glimpse of the context and motivation of my work. I present first the general field, and then the more specific domain of computational music analysis.

2.1 Short presentation and research areas

The terms *computer music* simply describe any activity that implies both music and computing tools.

Music and mathematics have been connected from origins. Links are numerous throughout history (see [2] for example). The basis of western music was developed by the Pythagoreans in the 5th century BCE. The musical stave[1] was invented during the 12th century, five hundred years before Cartesian coordinate systems. The French composer Jean-Philippe Rameau used mathematical tools to theorise harmony in the 18th century, in [23] for example. Modern composers such as Messiaen, Schönberg, or Xenakis[2] investigated new forms through mathematics. Sound is a physical vibration, and has mathematical properties; and western music is structured at every level by numbers. Hence, the birth and development of computer music naturally quickly followed the one of computers. The

[1] In French: *portée*.

[2] See for instance Xenakis, I. (1992). *Formalized music: thought and mathematics in composition* (No. 6). Pendragon Press.

progress of computation offered new tools to musicology; simultaneously, the all-analogue world of audio became almost all-digital.

Computer music is therefore a young science (the first pieces composed with the help of computers appeared in the late 50s), based on a very ancient one. In computer sciences, it is related to computational linguistics; it can be close to cognitive sciences and often uses techniques from machine learning.

Its research areas are numerous and diversified. One can deal with audio data (recordings, live performance...) or symbolic one (MIDI scores...); one can study existing pieces or aim at producing new ones; one can improve the laypersons' or the professional musicians' experience. Here is a non-exhaustive list of research areas being part of computer music:

- automated composition or orchestration;
- automated live improvisation;
- computational music analysis;
- music representation;
- signal processing...

2.2 Computational music analysis

Why analyse music with computers? As stated above, computers provide researchers with brand new ways of studying music. My analysis, for example, requires too many computations for a human being, and on the other hand no peculiar musical skills. And what kind of *analysis* can computers achieve? Anything that can help the work of musicologists will be called an analysis; the most common ways to do so are pattern discovery (as in the analysis that follows) and segmentation into meaningful units.

An analysis can be of audio material or of symbolic representation (or both); its goal can be to acquire knowledge or the ability to create new pieces (or both). The analysis I give here focuses only on symbolic data and its purpose is rather a learning one.

At the beginning of my internship, my work was mainly bibliographical, in order to become more familiar with the diversity of existing techniques for computational music analysis. A reader who would like to do the same may want to read papers using probabilistic grammars ([1]), Markov chains and n-grams ([20], [21], [4]), geometrical patterns ([18]), or tries (prefix trees) ([15]).

3 Analysing jazz chord sequences

In this section, after a presentation of the data motivating this project, jazz lead sheets, I introduce the main tools I used, namely compression algorithms chord similarity measures. Finally I present and discuss my results.

3.1 Motivation

The project was mainly motivated by an extract of a dataset of jazz lead sheets. This dataset was provided by Sony CSL, a research laboratory located in Paris of Sony Corporation, one of the world leading companies in audio technology. Data are an important issue in computer music, because most musical materials are not digitized yet, and databases are hard to gather. My task was to provide an analysis of this dataset.

Before going into more details, I have to introduce basic musical definitions. I try here to give enough explanations for anyone to understand what I will deal with, but also clues for a more interested or knowledgeable reader who would like to get a glimpse of mathematical models of music.

In jazz music, songs are displayed in the form of *lead sheets*, scores giving the base melody with extra indications of chords. An extract of a lead sheet for *What a wonderful world* by Louis Armstrong (1967) illustrates this in Figure 1; the sung melody is written on a stave and the chords are visible in handwritten font.

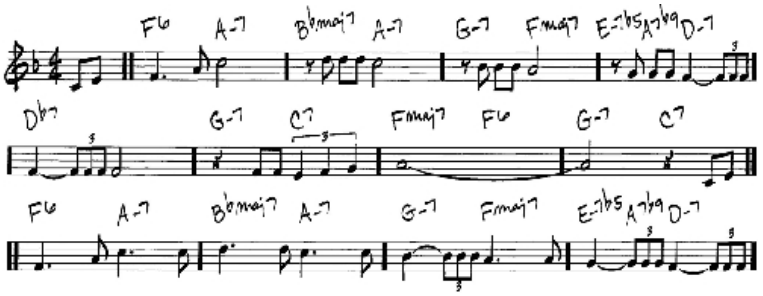

Figure 1: Lead sheet with the beginning of Louis Armstrong's *What a wonderful world*.

A *chord* is a set of at least three notes; in the jazz context, they are given as an indication of the current atmosphere of the piece, and musicians can improvise the accompaniment of the melody with the notes of the chord. Chords are represented[3] by a letter[4] from A to G with a possible *accidental* ($\#$ for *sharp*, $\flat$ for *flat*, or nothing), representing the *root note*, and a textual information giving the other notes relative to the root note. For example, if we consider the $C\#m7$ chord ("C sharp minor seventh"),

[3] There are several ways of representing chords; this representation, sometimes called the *tabular notation*, is the most common in popular music (jazz, pop, rock...). However, superposed notes on a classical music stave would also represent a chord.

[4] The French equivalent is to give names to notes, from *do* (C) to *si* (B).

the notes will be the root note, $C\#$, its *minor third*, E, its *fifth*, $G\#$ and its *minor seventh*, B. There are 313 different such textual informations in the Sony dataset[5].

The dataset I worked on contains the lead sheets of 30 jazz pieces by famous artists like Louis Armstrong, Billie Holiday or Charlie Parker. It is an extract of a larger set gathered by Sony CSL. I used only the files describing the chord sequences[6]. They were written in plain text format, and contained in all 1469 chords, given in the previous representation. The sequence below shows an example of how this data was encoded:

A Child Is Born: $B\flat M7;\ E\flat m;\ B\flat M7;\ E\flat m6;\ B\flat M9;\ E\flat m;\ A\ halfdim7;\ D7\#9\ldots$

So, the motivation for my work was to provide an analysis of this data. The purpose of such an analysis is to gain knowledge about jazz and chord sequences, understanding it better, and also in a second time to use this knowledge to be able to compose similar music that would fit in the corpus. Since I wanted an analysis of the entire data rather than analyses of each piece, I used as an input for the algorithms described in 3.2 a concatenation of the 30 songs: the sequence of all the 1469 chords.

3.2 Compression algorithms

The words "analysis" and "compression" will be used here with very close meanings. Indeed, compressing a piece means exhibiting its structure and separating the essential from the redundant. In other words, the analysis I did consisted in looking at how jazz chord sequences can be compressed, and interpreting the results of compression processes as new knowledge on jazz music. Hence, my analysis of the jazz lead sheets dataset will be described in terms of compression, and evaluated as such.

There are two complementary ways to approach computational music analysis[7]. One sees a piece as a linear sequence of notes (or chords, etc.), as the listener does: she can remember everything she heard but has no way to know what is coming until she actually hears it. The other approach views the piece from above, completely. In terms of data structures, the former considers the piece as a linked list and the latter as an array. "Linear" analyses use for example Markov models and Shannon's entropy; "global" analyses use for likely formal grammars and Kolmogorov's complexity.

The compression method I use simply describes a sequence of chords through the patterns (repeated sub-sequences) it contains, thus removing the redundancy. I designed, implemented and tested algo-

[5] With the model of section 3.3.1, one could theoretically form 2037 different chords for a given root note (number of subsets of size at least 2 of a set of size 11); the actually used chords correspond to the "harmonious" combinations, which is a subjective notion and thus depends on the music genre.

[6] Other files described the melodies (with pitch, onset and duration).

[7] They are of course also relevant in a wider context.

rithms for both approaches (linear and global), in order to be able to compare them. For the former, I used the classical algorithm known as LZ77; for the latter, a "diagonal pattern decomposition". They are presented in this order.

3.2.1 Lempel-Ziv 77 (LZ77)

The first algorithm (for the linear approach) is LZ77. It was designed by A. Lempel and J. Ziv in 1977, introduced in [28], and is a very popular compressing algorithm. I chose it because it is simple and efficient (most dictionary coders are based on it).

It takes as an input a list (so, a stream) of data and produces as an output a list of triples of the form (a, b, D) meaning: "go back a times, copy the next b data, and add D". In the original algorithm, restrained buffer and preview size are given as parameters; however, I decided to let them be unbounded in order to focus on the best possible results. The obtained algorithm is written in Algorithm 1, and a complete illustrative example of an execution is given in the appendix.

Algorithm 1: LZ77

Input: Queue of Chords $\texttt{I} = (C_1, \dots, C_n)$.

Output: Queue of triples $\texttt{L} = (a_j, b_j, C_{i_j})_j$.

Begin

- $\texttt{buffer} \leftarrow$ empty queue
- **While** $\texttt{I}$ *is not empty* **do**
  - $\pi \leftarrow$ longest prefix of $\texttt{I}$ in $(\texttt{buffer} \cdot \texttt{I})$, beginning in $\texttt{buffer}$
  - $\mathsf{a} \leftarrow \texttt{size(buffer)} -$ (beginning index of π (in $\texttt{buffer}$)) (0 if none)
  - $\mathsf{b} \leftarrow$ length of π (0 if none)
  - **For** j **from** 1 **to** b **do**
    - $\texttt{buffer.push(front(I))}$
    - $\texttt{I.pop()}$
  - $\texttt{L.push}(\mathsf{a}, \mathsf{b}, \texttt{front(I)})$
  - $\texttt{buffer.push(front(I))}$
  - $\texttt{I.pop()}$
- **Return** $\texttt{L}$

With a computation of the longest prefix in $\mathcal{O}\left(|\texttt{I}|^2\right)$, the overall complexity is $\mathcal{O}\left(|\texttt{I}|^3\right)$. There are more accurate evaluations of this complexity, discussed among other in [28], though not of much interest here. Knowing that the complexity is polynomial and that the implementation runs fast is enough.

3.2.2 Diagonal pattern decomposition

The second algorithm (for the global approach) uses *diagonal patterns*. The best way to understand this notion is to visualise it. For a given piece, we write the chord sequence both vertically and horizontally. We can then consider a matrix whose cell (i, j) will correspond to the ith and jth chords of the input sequence. Let us draw this cell in white if these chords are identical, and in black if they are different. We get a binary matrix by which we can see the diagonal patterns, which are diagonal sequences of white cells. They correspond to sequences of chords that appear (at least) twice in the input piece, on two different positions. Figure 2 shows such a matrix (for an extract from *Giant Steps*, by John Coltrane).

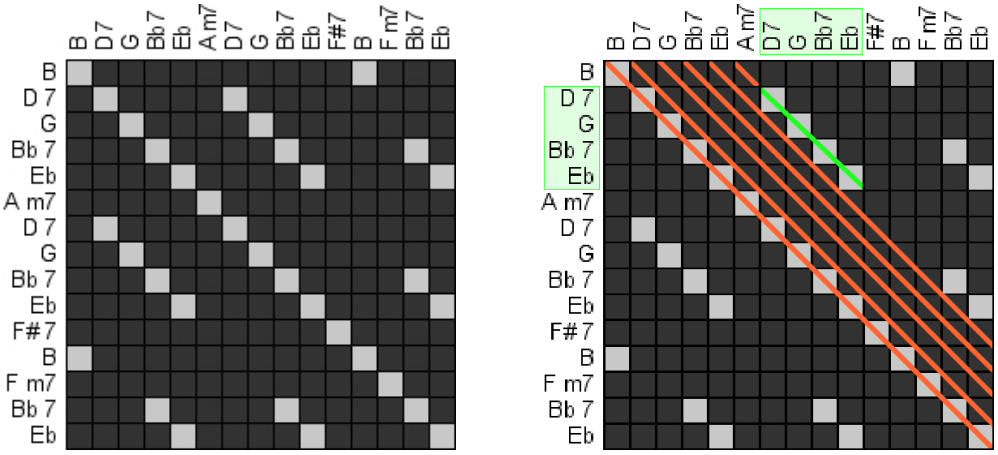

Figure 2: A binary matrix and the diagonal search for patterns.

The *patterns* selected by the algorithm precisely are maximal diagonal sequences, along with their occurrences (positions where they occur). *Maximal* means that we consider only sequences having an occurrence that cannot be extended. Figure 2 shows the diagonal search for patterns and the maximal pattern $D7; G; B\flat7; E\flat$. $D7; G$ is not maximal: every times it occurs it can be extended in the end. On the contrary, $B\flat7; E\flat$, which is included in $D7; G; B\flat7; E\flat$, is also a maximal pattern, occurring in the very end. In the piece of Figure 2, the diagonal patterns are (by increasing size):

- B, at positions 0 and 11;
- $B\flat7; E\flat$, at positions 3, 8 and 13;
- $D7; G; B\flat7; E\flat$, at positions 1 and 6.

Once the list of patterns is established, the next step is to select a subset of these patterns which is sufficient to describe the whole piece. It can happen, like in the current example, that it not possible to completely cover the input piece with the diagonal patterns. To tackle this issue, we add a "pattern" for each single chord appearing in the piece (here: B at positions 0 and 11, $D7$ at position 1, G at position 2...). The selected subset should be as "light" as possible, the *weight* of a pattern being the sum of its length and of its number of occurrences. This problem is a *weighted set cover*, which is

NP-complete[8]. So I implemented two heuristics, and the lighter results of the two is selected. Briefly, they are both greedy algorithms; one aggregates patterns until covering the whole piece and the other removes as many patterns as possible from the exhaustive list. Their pseudo-codes are given in the appendix.

The complexity of the whole algorithm is $\mathcal{O}\left(|\mathtt{I}|^5\right)$ [9]. This is quite high. In practice, the number of maximal patterns (which is $\mathcal{O}\left(|\mathtt{I}|^2\right)$ in the worst case) seems to be the most determining factor for the running time. On my complete database (approximately 1500 chords), the execution takes between a few seconds and several minutes.

3.3 Similarity measures for chords

This section introduces the concept of chord similarity measures, and how I used them to improve the algorithms from previous section. Specific similarity measures have been developed for chords, to quantify formally their audible differences. I define here a mathematical model of chords, and then the state-of-the-art measures I used. Finally I present their incorporation into the algorithms and its consequences.

3.3.1 A mathematical model of chords

There are only 12 possible root notes because two notes separated by an *octave* (concretely, whose frequencies ratio is a power of two) sound the same, and are thus called the same: C refers to low-pitched as well as to high-pitched sounds. The 12 notes correspond to a division of the octave into twelve equally spaced *semitones*; this is called the "well-tempered" scale. So, as shown in Figure 3, the keyboard can be mapped to the cyclic group $\mathbb{Z}/12\mathbb{Z}$, and the notes to integers from 0 to 11[10].

Figure 3: A keyboard with note names and their mapped values in $\mathbb{Z}/12\mathbb{Z}$.

Chords can then be seen as subsets of $\mathbb{Z}/12\mathbb{Z}$. For instance, the previously seen $C\#m7$ chord would be mapped to the set $\{4;7;11;2\}$ [11]. Moreover, the harmonic content in the chord name (here, "$m7$")

[8] For more explanations on the set cover problem, see [14] or [5].
[9] See appendix for a brief analysis.
[10] Usually, C is mapped to 0; it has no importance here and I chose to map A to 0 instead, which seemed simpler.
[11] The order can be important, but does not matter here.

can be seen as a vector and the root note ("$C\#$") as a starting point. Chords with same labels but different root notes are then the same by transposition: $C\#m7$ corresponds to $\{4;7;11;2\}$ and $D\#m7$ to $\{5;8;12;3\} = \{4;7;11;2\} + \{1;1;1;1\}$.

It is hence possible to define formal distances between chord. However, such a distance must be chosen carefully to be relevant. Canonical distances do not necessarily mirror what is heard. With the "city block distance") on chords of three notes, the chords $C = \{3;7;10\}$ and $Cm = \{3;6;10\}$ would be close since the difference is only of one semitone; but to the ear they sound very different, one being major and the other minor. Likewise, $C = \{3;7;10\}$ and $Bm = \{2;5;9\}$ would sound closer than they may look: they are "inversions" of each other.

3.3.2 List of used measures

In all, I used 10 different measures (including equality), some of which are rather elementary, and some specifically developed for this field. Three take as an input the two chords to compare, C_1 and C_2, and return a boolean (`true` if and only if the chords are similar). All three define equivalence classes. They are:

- ▷ root note equivalence: `true` iff C_1 and C_2 have the same root note;
- ▷ transposition[12] equivalence: `true` iff C_1 and C_2 have the same harmonic indication (but possibly different root notes);
- ▷ PCS-Prime equivalence (see [9]): `true` iff C_1 can be obtained from C_2 by a combination of inversions and transpositions.

The six other measures are distances[13]: they take as an input two chords C_1 and C_2 and return a positive real number. As I will show in the next part, this value was needed only to determine if two chords are "close" or "different", so I used them with a threshold as additional input parameter. The measure will then return `true` if and only if the distance between C_1 and C_2 is less than the threshold.

They are the F1-score (cardinality of the intersection divided by the sum of the cardinalities, multiplied by two); Eric Isaacson's similarity index, defined in [12]; David Lewin's measure, defined in [16]; Robert Morris' measure, defined in [19]; John Rahn's measure, defined in [22]; Richard Teitelbaum's measure, defined in [27].

The definitions of all these measures can be found in the appendix. Here I will only present as an illustrative example Isaacson's similarity index. It is a relatively arbitrary choice since, as I will show

[12] The word "transposition" in music correspond to the geometrical notion of "translation".
[13] I use the word "distance" to emphasize the difference with the first three measures; however all are not distances in a mathematical meaning.

in section 3.4, all these measures gave close results. Nevertheless, Isaacson's similarity index allows me to introduce the interesting notion of interval vector.

Given a set $S \subset \mathcal{P}([0,11])$, its associated *interval vector* is a vector $IV(S) \in \mathbb{N}^6$ such that the i-th coordinate of $IV(S)$ is the number of pairs of elements in S whose difference is i. In other words, we enumerate the intervals inside S: how many intervals of length 1, of length 2, etc. The interval vector is a histogram for these values. There are 6 and not 12 coordinates because these intervals are not oriented (modulo 12, going from 1 to 8 is the same as going from 8 to $12+1=13$, and the interval has then a length of 5).

For example, for a major seventh chord, whose corresponding set is $\{0;4;7;10\}$, we have 1 interval of length 2 (between 0 and 10), 2 of length 3 (4 and 7, 7 and 10), 1 of length 4 (0 and 4), 1 of length 5 (0 and 7) and 1 of length 6 (4 and 10). Hence $IV(\{0;4;7;10\}) = (0,1,2,1,1,1)$.

Isaacson's similarity index is defined as the *standard deviation* function applied to the interval vectors. For chords X and Y with respective interval vectors $IV_X = (x_1,\ldots,x_6)$ and $IV_Y = (y_1,\ldots,y_6)$, let us denote by D the difference vector $((y_1-x_1),\ldots,(y_6-x_6))$ and $\bar{D}$ the mean of the (y_i-x_i)s. The measure is finally:

$$\mathfrak{M}_{Isaacson}(X,Y) = \sqrt{\frac{1}{6}\left(\sum_{i=1}^{6}\left(D_i-\bar{D}\right)^2\right)}$$

3.3.3 Improvement of the compression algorithms

There exist *lossy* and *lossyless* compression algorithms. *Lossyless* means that the decompression of the compression is equal to the original sequence, while it is not necessarily so with *lossy* algorithms. The LZ77 algorithm and the diagonal patterns algorithm both perform a compression without loss. I transformed them into lossy algorithm; this allows better compression, i.e, smaller outputs. Again, the compression of jazz chord sequences is intended as an analysis. This is why I chose specific measures to musical analysis: improving the compression with meaningless measures would not be relevant.

The loss is introduced by replacing equality tests in the algorithms by similarity tests. In the LZ77 algorithm, equality tests occur when computing prefixes. Instead of the longest common prefix, the longest similar (for a given measure, up to a certain threshold if needed) prefix is selected. In the diagonal patterns algorithm, equality tests occurs when computing the binary matrix. Instead of drawing a cell of the matrix in white only if the two corresponding chords are equal, this will be done also if they are similar (for a given measure, up to a certain threshold if needed). The matrix has then more white cells, implying more and longer patterns. Figure 4 shows this transformation (with the F1-score and a threshold of 0.9).

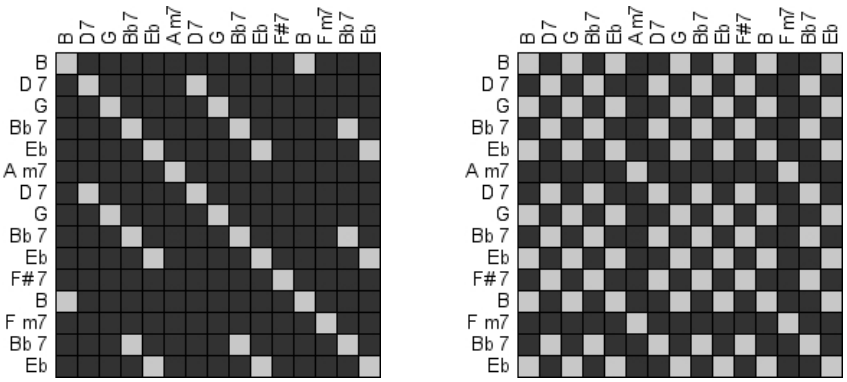

Figure 4: Binary matrices for *Giant Steps*, the second using a similarity measure (F1 score).

The compression processes are evaluated with their *compression factor* and *recovery factor*.

The *compression factor* corresponds to the size of the input data divided by the size of the compressed data. It is expected to be as high as possible (and at least greater than 1). I consider that the size of a chord is the same as the size of an integer. Indeed, they are $7 \cdot 4 \cdot 313 = 6573$ possible chords, so a chord can be described with $\lceil log_2(6573) \rceil = 13$ bits; and the integers dealt with are lesser than 1469 (the total number of chords in the database), thus defined by $\lceil log_2(1469) \rceil = 11$ bits. Of course, this definition fits the data I used and a different one could be needed for a different dataset.

The *recovery factor* is the ratio of chords in the decompression of the compression which are equal to the corresponding chords in the input piece. It is thus a real number between 0 and 1, that we will expect to be as close to 1 as possible. This definition is rather sensitive. Indeed, the loss is introduced by the similarity measures: input chords are replaced during the compression process only by similar ones. The only other way to define the recovery factor would be to use precisely the similarity measure ("the two corresponding chords are different by at most t according to the measure"), however, in my eyes, not having an external evaluation would mean having too much faith in the measures. Moreover, while one understands well what it is to be equal or different, it is not clear what a difference of t for a given precisely means.

So a decompression algorithm is needed to compute the recovery factor. It is very simple for the LZ77 algorithm. For the diagonal patterns algorithm, the covering patterns are sorted by increasing number of occurrences[14] and are copied in this order (so, a position that is covered by several patterns is rewritten several times, and only the last written chord, coming from the pattern with most occurrences, will be kept). A pre-treatment is done on the patterns in order to maximize the recovery factor. Indeed, a pattern occurring *similarly* several times can correspond to several different exact sequences, and the recovery factor will depend on the choice of the sequence (which does not impact the compression factor) representing the pattern. The pre-treatment simply consists in computing

[14] This sorting criteria has been chosen empirically, because it led to the best recovery factors. Yet, is not clear why it should be better.

for every pattern what positions of the reconstruction depend on it, and choose among the possible sequences the one that implies a minimum loss[15].

3.4 Results

The two algorithms have been run on the dataset of 1469 chords with all the measures and several thresholds. This section presents the results obtained by comparing the compression and recovery factors obtained by these compression processes. I focus here on their interpretation; more exhaustive numerical results can be found in the appendix. First the different measures are compared, and then the two algorithms.

Comparison between measures

The 10 different measures I used resulted in different compression results. For different measures, binary matrices look differently; recovery and compression factors as functions of the thresholds look differently (as can be seen in the appendix). However, surprisingly, there is a correlation between these two factors, independent from the measure used, as can be seen in Figure 5, where . So, as one would have expected, there is a trade-off between the compression and the recovery factors: the better the compression, the higher the loss. Also, this feature validates the use of the F1-score, which in contrary to other measures was not designed specifically for music analysis, in this field.

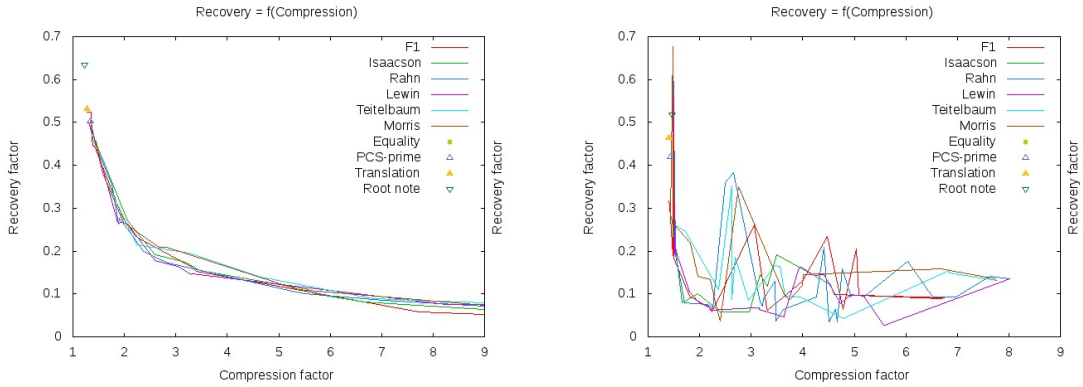

Figure 5: Link between recovery and compression factors (left: LZ77, right: diagonal patterns).

In the case of the LZ77 algorithm, the results are really the same. Curves in Figure 5 (left) have very close shapes. As for the diagonal patterns algorithm, some measures are “leading” around several values of compression factors. But it is hard to determine if it comes from the irregularities of the algorithm or really originates from some kind of superiority. Roughly, leading measures are

[15] This greedy algorithm, knowing the decompression scheme, finds obviously the optimal configuration of this problem.

Teitelbaum's for compression factors between 1.5 and 2.5, Morris' between 2.5 and 3.5, Isaacson's similarity index between 3.5 and 4, the F1-score between 4 and 4.5, and again Morris' above 4.5.

As mentioned above, I use the recovery factor in order to benefit from an evaluation instrument that is exterior to the measures. Nevertheless, differences between them appear if we are confident in their ability to reflect musical closeness between chords. Indeed, in the case of the diagonal patterns algorithm, the best compression factors are not obtained for the same thresholds. All lead to highest compressions for medium thresholds, except for Isaacson's similarity index, shown in Figure 6: the compression factor is almost an increasing function of the lossiness. This could be fortuitous, or mean that this precise measure is particularly adapted to the analysis of jazz chords. The best way to find out the truth would be to have the opinion of a skilled musician, who for instance would listen to the original piece and the decompression of the compression.

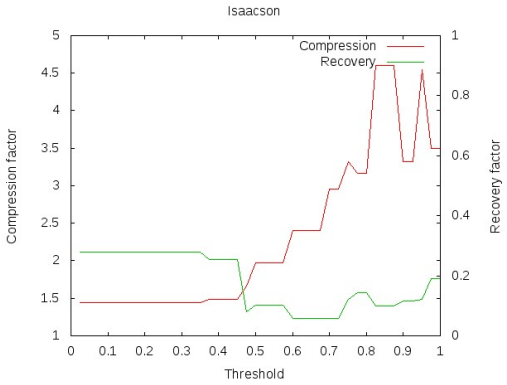

Figure 6: Diagonal compression with Isaacson's measure.

The relevance of similarity measures

The measures with no thresholds result in relatively low compression factors (and high recovery factors). The lossiless version of the LZ77 algorithm does not even compress: the compression factor is $0.931258 < 1$. The gain provided by the use of similarity measures becomes obvious with the ones using thresholds. They make possible a wide range of compression factors.

The fact that "loosening" the chord sequence actually improves the compression process brings to light the existence of a hidden structure of the pieces. It reveals that under an irregular surface there is a more simple shape. Using similarity measures smooths the surface and simplifies the analysis.

Comparison between algorithms

The LZ77 algorithm produces much higher compression factors than the diagonal patterns algorithm. This is not surprising, as for an input size $|\mathtt{I}|$, the shortest possible output of LZ77 is of constant

length 2: $(0, 0, \mathtt{I}[1]), (1, |\mathtt{I}| - 2, C)$. For the diagonal patterns algorithm, the shortest possible output is of length $\sqrt{|\mathtt{I}|}$: one pattern of length $\sqrt{|\mathtt{I}|}$ occurring $\sqrt{|\mathtt{I}|}$ times[16]; and none of the implemented algorithms for the set cover problem would choose this configuration if all the chords of the input are similar.

The difference between the algorithms in terms of achieved compression factors is thus not very relevant. However, it is interesting to note that, for the measures without thresholds, the diagonal patterns algorithm obtains better compression factors (and lower recovery factors) than LZ77. For instance, when using the root note equivalence, the compression factor of LZ77 is 1.22421, while it gets to 1.46531 with the diagonal patterns algorithm (full results can be seen in the appendix).

Furthermore, one observes in Figure 5 that the results of the diagonal patterns algorithm are a lot more "chaotic" than the results of the LZ77 algorithm. Those are very regular: the compression factor is almost always a decreasing function (and the recovery factor an increasing function) of the similarity, represented by the threshold. One reason for the irregularities in the diagonal patterns algorithm is obviously the approximations made when solving the set cover problem[17]. Moreover, it is not clear like (as it is for the LZ77 algorithm) that loosening the threshold should improve the compression. For instance, there can be a pattern appearing several times such that, after the loosening, some of its occurrences are extended and some are not; we would then have two patterns or more, and the sum of their weights would be greater than the weight of the single original pattern.

The second important observation is that for the same compression factors, the recovery factor is generally higher (except around 2) with the diagonal patterns algorithm. In Figure 5, one sees that for a compression factor of 2.5, the recovery factor obtained by the LZ77 algorithm is around 0.2, while with the diagonal patterns algorithm it can be up to almost the double: 0.4. This can reflect that the second paradigm of analysis (the "view from above", opposed to the "linear" approach) better fits jazz chord sequences[18].

[16] The optimal compression is obviously composed of a single pattern; if it is of length $|\pi|$ it has to occur $\frac{|\mathtt{I}|}{|\pi|}$ times, and its weight is then $|\pi| + \frac{|\mathtt{I}|}{|\pi|}$, with a minimum for $|\pi| = \sqrt{|\mathtt{I}|}$.

[17] For the set cover problem without weights, the greedy algorithm similar to one I use performs a H_n-approximation, H_n being the n-th harmonic number (for n the size of the problem); this is of the order of $\ln(n)$.

[18] One should be cautious, of course, since the LZ77 algorithm is not as complex; on the other hand I stated that the diagonal patterns algorithm could be improved.

4 Other and further work

In this section I present ways I can improve my contribution, and my contribution to the Lrn2Cre8 project. All of this represents a non-negligible part of my work in Aalborg. I focused in this report on my most complete results; here I try to give hints to the rest, general ideas rather than details.

4.1 Segmentation and grammatical inference

The drawings of binary matrices used for compression (section 3.2.2), like Figure 7, reveals pretty geometrical configurations, symmetries, and hence structure. The compression described here is to capture this structure; but there surely are many different ones.

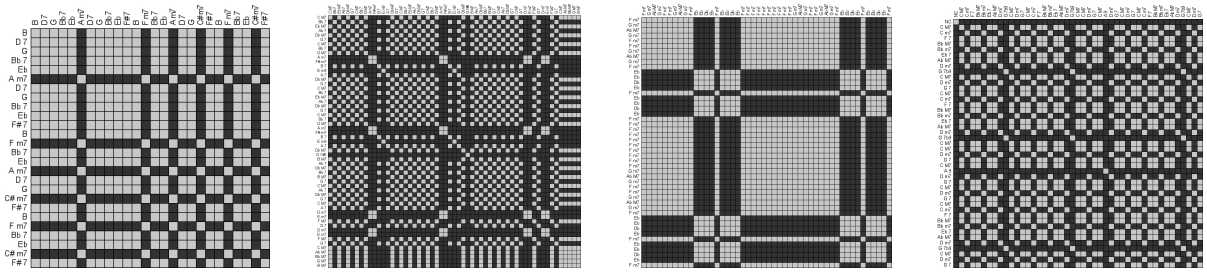

Figure 7: Several binary matrices revealing geometrical structure.

My first idea was to compute a segmentation of the piece using the matrix (segmentation is a common way to analyse a music piece, since it gives a global structure). One can often see what looks like a partition of the matrix and consequently of the piece. However, such partitions can appear for different measures, sometimes very different thresholds. They may not appear at all for some pieces, or look very different. So I did not manage to design an algorithm segmenting satisfyingly my data.

The second way that I have been much interested in is *grammatical inference* (or *grammar induction*). As presented in [1], formal grammars, that have been introduced for the modelling of spoken languages, have a lot of applications in computer music. Furthermore, a grammar that could generate a piece or a corpus provides both a synthetic analysis and a way of creating similar pieces.

The problem of grammatical inference is, given two sets S^+ and S^- (possibly empty) of words on an alphabet Σ, to compute a grammar of a certain form (regular grammar, context-free grammar...) that generates every word of S^+ but none of S^-. [8] is a good bibliographical introduction to the topic, and [10] and [11] a complete survey of state-of-the-art techniques. Several works use grammatical inference on a very close topic to mine: finding a structure inside jazz chords (not like me for a sequence of chords, but for the different chords used in jazz among all the possible combinations of notes) : [7], [24], [26], [13].

Nevertheless, I did not find a way of combining my work with these techniques.

In conclusion, I consider that what I did provides an interesting analysis of the data I was given, but that it also reveals potentials still untapped.

4.2 Improving the compression scheme

There would be many ways to improve the compression and especially the diagonal compression. Beyond those I already mentioned, I can imagine two important new ones.

The first would be to use information from the melodies as a complement to the chord sequences. There are of course many ways of (and many papers about) combining them. And it would be possible to do so since most of the songs from the database I worked on also contain informations about the melody.

The second would be to incorporate the works described in [3]. This papers looks for diagonal patterns (from an audio input), but uses techniques from the field of image analysis (like blurring, convolutions, Hough transform...) to identify more sequences as diagonal patterns, even is they are slight holes, or a different orientation (see Figure 8). I believe this could improve much the results of the algorithm I used.

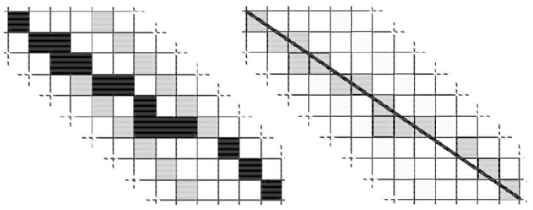

Figure 8: An "approximate diagonal pattern" (figure from [3]).

Moreover, since the results on the partial dataset of 30 songs are promising, the algorithms will be tested on the whole corpora, in order to see how the results generalize.

4.3 The Lrn2Cre8 project

The Lrn2Cre8 project[19] is a European project between six research institutes working on computer music (Aalborg University being one of them). Its purpose is to develop *learning* techniques of music in order to be able to *create* new pieces. One way consists in adapting to composition already existing analysis methods.

In this context, I discussed with Olivier LARTILLOT, Ph. D. at Aalborg University, who worked in the same group as I did, about the adaptation of his tool, PATMINR ([15]). While I was trying to use grammatical inference in my own work, it appeared to me that a technique could be combined with

[19] "Learn to Create", see [17] for more information.

his. Described in [6] and used in [25], it is designed to infer grammar for a specific data, that has been bracketed beforehand so as to emphasize its structure. Originally, this aimed at analysing natural languages, a bracketed data being for instance the sentence "[My neighbour [ate [his yoghurt] [with [a spoon]]]]".

It made me think of PATMINR, because the resulting decomposition of the melody it performs (Figure 9, on the left) can easily be transformed into a bracketing (Figure 9, on the right; colors are used for visibility but have no meaning). The challenging task, now, is to cleverly label the bracketed piece, before applying the grammar induction methods. This idea is still currently being investigated.

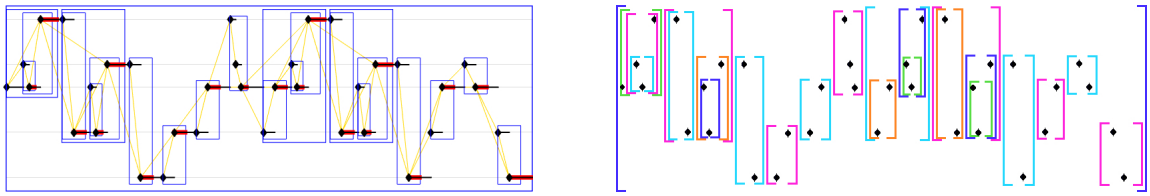

Figure 9: Analysis of a piece with PATMINR and resulting bracketing.

5 Conclusion

I have proposed two compression schemes for jazz chord sequences, which perform an interesting analysis of the data I had to study. They combine algorithms for compression without loss and similarity measures loosening the data. The global approach achieves better results than the linear one, showing general structure in jazz.

Moreover, this project brings forward the use of association measures for compression, which can be extended to many others analysis of chords; it asserts the usefulness of the F1-score in this context; and the binary matrices generated with these measures still show important potential for further investigations.

6 Bibliography

References

[1] Gold NE Abdallah SA. Comparing models of symbolic music using probabilistic grammars and probabilistic programming. In *Joint Sound and Music Computing Conference and International Computer Music Conference 2014*, 2014.

[2] Raymond Clare Archibald. Mathematicians and music. *American Mathematical Monthly*, pages 1–25, 1924.

[3] Jean-Julien Aucouturier and Mark Sandler. Finding repeating patterns in acoustic musical signals: Applications for audio thumbnailing. In *Audio Engineering Society Conference: 22nd International Conference: Virtual, Synthetic, and Entertainment Audio*. Audio Engineering Society, 2002.

[4] Darrell Conklin and Christina Anagnostopoulou. Representation and discovery of multiple viewpoint patterns. In *Proceedings of the International Computer Music Conference*, pages 479–485. Citeseer, 2001.

[5] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[6] Stefano Crespi-Reghizzi. The mechanical acquisition of precedence grammars. Technical report, DTIC Document, 1970.

[7] W Bas De Haas, Martin Rohrmeier, Remco C Veltkamp, and Frans Wiering. Modeling harmonic similarity using a generative grammar of tonal harmony. 2009.

[8] Colin De La Higuera. A bibliographical study of grammatical inference. *Pattern recognition*, 38(9):1332–1348, 2005.

[9] Allen Forte. *The structure of atonal music*, volume 304. Yale University Press, 1973.

[10] King-Sun Fu and Taylor L Booth. Grammatical inference: Introduction and survey-part i. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(SMC-5):95–111, 1975.

[11] King-Sun Fu and Taylor L Booth. Grammatical inference: Introduction and survey-part ii. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (3):360–375, 1986.

[12] Eric J Isaacson. Similarity of interval-class content between pitch-class sets: the icvsim relation. *Journal of music theory*, pages 1–28, 1990.

[13] Philip N Johnson-Laird. Jazz improvisation: A theory at the computational level. *Representing musical structure, London*, pages 291–325, 1991.

[14] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[15] Olivier Lartillot. Patminr: In-depth motivic analysis of symbolic monophonic sequences. *Music Information Retrieval Evaluation eXchange*.

[16] David Lewin. A response to a response: on pcset relatedness. *Perspectives of new music*, pages 498–502, 1979.

[17] Lrn2cre8 project homepage: http://lrn2cre8.eu/.

[18] David Meredith. Cosiatec and siateccompress: Pattern discovery by geometric compression. In *International Society for Music Information Retrieval Conference*, 2013.

[19] Robert Morris. A similarity index for pitch-class sets. *Perspectives of New Music*, pages 445–460, 1979.

[20] François Pachet, Pierre Roy, Gabriele Barbieri, and Sony CSL Paris. Finite-length markov processes with constraints. *transition*, 6(1/3), 2001.

[21] Marcus T Pearce and Geraint A Wiggins. Auditory expectation: the information dynamics of music perception and cognition. *Topics in cognitive science*, 4(4):625–652, 2012.

[22] John Rahn. Relating sets. *Perspectives of New Music*, pages 483–498, 1979.

[23] Jean Philippe Rameau. *Nouveau systême de musique théorique, où l'on découvre le principe de toutes les règles nécessaires à la pratique: pour servir d'introduction au Traité de l'harmonie*. De l, 1726.

[24] Martin Rohrmeier. Towards a generative syntax of tonal harmony. *Journal of Mathematics and Music*, 5(1):35–53, 2011.

[25] Yves Schabes, Michal Roth, and Randy Osborne. Parsing the wall street journal with the inside-outside algorithm. In *Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics*, pages 341–347. Association for Computational Linguistics, 1993.

[26] Mark J Steedman. A generative grammar for jazz chord sequences. *Music Perception*, pages 52–77, 1984.

[27] Richard Teitelbaum. Intervallic relations in atonal music. *Journal of Music Theory*, pages 72–127, 1965.

[28] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.