Statistical Analysis of Music Corpora

TOPPS · Semantics-based Program Analysis and Manipulation

January 16, 2009

Johan Sejr Brinch Nielsen

Email:	zerrez@diku.dk
Cpr.:	260886-2547
Supervisor:	Jakob G. Simonsen

Dept. of Computer Science,
University of Copenhagen

Abstract

I investigate trends in musical complexity, specifically by computing the entropy of chord sequences.

I analyse 571 Mozart pieces (of 626 in total[1]). The result shows an increase in entropy over time in at least five categories, specifically *divertimentos and serenades*, *piano pieces*, *piano trios*, *string quartets* and *symphonies*. And increase in entropy is also observed as the works grow larger; the correlation between chord mass and entropy is shown to be 0.59. Furthermore, I show that operas generally has the highest entropy (2.47), while canons and songs has the lowest (0.86 and 0.74 respectively).

The results are based on Mozart works only, thus only applies to these. However the framework used to compute the results is general and can be reused or expanded in future studies.

The results show promise for entropy as a measure for musical complexity.

[1] Mozart works categorized by L. von Koechel (www.classical.net)

Contents

Chapter 1

Introductory Theory

1.1 Introduction

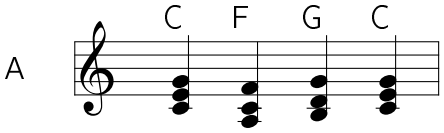
1.2 Statistical Methods

1.3 Identification of Chords

1.1 Introduction

Musical pieces are built on a foundation of chords, which are selected from the Circle of Fifth (Miller, 2005); a diagram that explains which chords sounds *well* together. As a consequence of deriving chord progressions from this system the musical pieces end up sounding harmonic, however alike.

An example of a chord progression that appears in a countless number of songs are (see Williams, 2008):



The goal of this project is to discover these typical chord progressions generically to measure the complexity of the piece itself. I use information entropy from basic statistics to model this complexity. This model supports history of size n, which I exploit to detect the chord progression repetitions.

I discuss how musical information can be obtained, processed and prepared for the statistical analysis. This process includes generation of MusicXML from graphical notation formats (e.g. Bitmap images) and extraction of the needed information from MusicXML.

I create an environment that allows for such analysis of complexity in an automated manner. Using this environment I examine how the complexities of 571 of Mozart's works develop over time and category.

1.1.1 Contributions

I show how the complexity of musical pieces can be described using entropy and perform statistical analysis on 571 of Mozart's works, spreading across 32 categories.

I show that the entropy values of Mozart works grow year by year in at least 5 categories, specifically *divertimentos and serenades*, *piano pieces*, *piano trios*, *string quartets* and *symphonies*. I show that this is also the case across categories during Mozart's first 16 years. I show that operas are most likely to have a high entropy value, whereas finding such in canons and songs are least likely. Furthermore, I show that longer works have higher entropy values.

Furthermore I provide a framework that implements generation of MusicXML from graphical notation formats (e.g. BMP files), extraction of necessary musical information from MusicXML and analysis of such using entropy. The modular design of the framework eases it's use in future work and allows it to be easily expanded.

1.1.2 Related Work

(Shannon, 1948) introduced the field of information theory and formalised the models for both conditional and unconditional entropy.

(Margulis, 2005) proposed a model of melodic expectation, based on heuristics rather than entropy. The project attemps to quantify the musical experience of a work, based on a small selection of progression features.

(Temperley, 2007) discusses how music and probability can be united using a Bayesian. Musical elements such as pattern perception, harmony, improvisation, and musical styles are discussed.

(Margulis and Beatty, 2008) uses entropy to compare the complexity of works. The materials used are spread over several artists (including Mozart and Bach), however significantly smaller than what is used in this project (a total of 167 works). Their model compares 8 different features, including active parts and note lengths, but excluding chord progression which is analysed in this project. Furthermore, their model uses a history of 1 where the model I present in this paper generalises this to history of n.

1.1.3 Expectations of the Reader

I assume that the reader has an understanding of musical notation, including the meaning of notes, measures and parts, corresponding to the level of (Surmani, 1998). Besides this, the reader should know of the XML file format and how this format is used to structure other formats, corresponding to the level of (Ray, 2003). The reader should also have knowledge of basic math and statistics, corresponding to the level of (Johnson, 2008).

1.1.4 Overview

In Section 1.2 I explain how entropy is a representation of musical complexity. I introduce the statistical methods needed when working with entropy and show how unconditional entropy can be generalised into conditional entropy.

In Section 1.3 I give a recursive algorithm for identifying an accord based on its note.

In Section 2.1 I give an overview of the implementation details.

In Section 2.2 I compare the International Music Score Library Project with Mozarteum, with respect to automated computing.

In Section 2.3 I go through the work involved when converting scores to the MusicXML format. I describe several different OCR applications and why I ended up using SharpEye.

In Section 2.5 I describe the details of the MusicXML format. I describe how MusicXML represents notes, measures and parts and document where MusicXML differs from typical XML formats.

In Section 2.6 I describe the problems involved with scanned notes, and give a simple algorithm for improving such notes.

In Section 3.1 I describe the problem of interpolation and show how this problem can be solved using operations research. I move on to describe the correlation coefficient and how this can be computed. Afterwards I go through the statistical results.

1.2 Statistical Methods

The goal of this project is to compare the complexity of musical works. As a measure of complexity I use predictability; how easy the work can be predicted. The complexity of a work is then the probability of guessing the next chord in the work, knowing the n preceding chords.

This probability of guessing the next chord is computed from the number of possible choices,N, for each chord:

$$p = \frac{1}{N}$$

The number of possible choices is based on the current work and how chords are used in this particular work. Since the probability is computed from N, only N is needed when comparing predictability. I go even further and say that only the number of digits used to represent N is needed. The number of digits in N is computed as:

$$D = log_b N$$

where b is the chosen base. The higher the number of digits needed to represent the possible choices, the lower the predictability.

The number of digits needed to represent any state of a sequence can be computed using statistical entropy. To reach the probability of guessing the next chord, I compute the entropy value, H, and then the probability by:

$$p = \frac{1}{b^H}$$

In this project I will focus on digits in base 2 (bits), a natural choice in the field of Computer Science.

1.2.1 Statistics and Music

Before describing the statical methods I use in this project, I introduce the needed terminology:

- *Random variables*: variables who's values are unknown
- *Random Vector*: a vector containing several random variables
- *Events*: occur when a random variable takes a particular value

- *Probability spaces*: defines all possible outcome of the variables together with their probability

I use a random variable to represent a chord. A work is then a sequence of successive experiments with a single random variable. The probability space contains all possible chords, chosen as:

- All standard chords, major or minor:
  "A", "Am", "Bb", . . . , "Gb", "G"
- All standard chords expanded with the seven:
  "A7", "Am7", "Bb7", . . . , "Gb7", "G7"

More complex variations of chords (such as adding the second or six), are not counted as begin different from the standard chord.

1.2.2 Entropy

The entropy value measures the information contained in the random vector, hence how many bits[1] are needed to represent the vector, no matter state:

The entropy value of a random vector, X, is computed as:

$$H(X) := \sum_{i=1}^{|E|} P(E_i) \cdot \log_2(\frac{1}{P(E_i)})$$

$$= -\sum_{i=1}^{|E|} P(E_i) \cdot log_2(P(E_i))$$

where E is the set of possible events in the probability space and $P(e \in E)$ is the probability of event e. The probability of an event is computed as its frequency:

$$P(e) := \frac{\mid \{e' = e, e' \in E\} \mid}{\mid E \mid}$$

The highest entropy value occurs when all events are equally likely, while the lowest entropy occurs when just one event occurs is repeated. This is what one would expect from predictability.

The highest possible value is ($P(E_i) = \frac{1}{|E|}$):

$$-\sum_{i=1}^{|E|} P(E_i) \cdot \log_2(P(E_i))$$

[1] Base 2 is used, as described in Section 1.2

$$= - \mid E \mid \cdot \frac{1}{\mid E \mid} \cdot \log_2(\frac{1}{\mid E \mid})$$

$$= - \log_2(\frac{1}{\mid E \mid}) = \log_2(\mid E \mid)$$

While the lowest is $(P(E_1) = 1, P_{i>1} = 0)$:

$$\sum_{i=1}^{|E|} -1 \cdot P(E_i) \cdot \log_2(P(E_i))$$

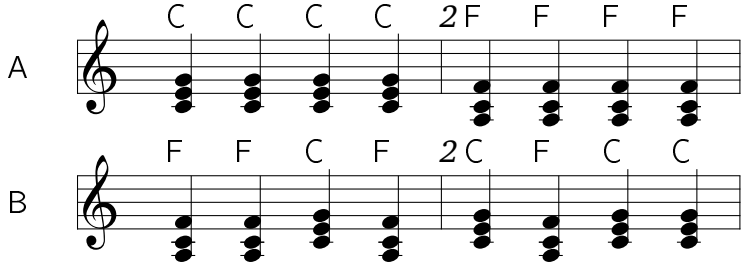
$$= -P(E_1) \cdot \log_2(P(E_1)) - \sum_{i=2}^{|E|} P(E_i) \cdot \log_2(P(E_i))$$

$$= -1 \cdot 0 - \sum_{i=2}^{|E|} 0 \cdot \log_2(0) = 0$$

These extrema values will be constant despite the actual musical work, since the number of possible chords will always be constant.

This computation does however have an unfortunate property. Since the entropy value is independent of the order of the events in S, any permutation of S will yield the same entropy value (the same predictability).

An example of this could be the following two musical pieces:



Piece A would sound very monotonic, since it repeats four chords at a time. B seems to variate its use of the two chords more than A and should therefore be awarded with a higher entropy value.

However, the above definition of entropy yields the same entropy value for both pieces, since B is a permutation of A.

I describe how this problem is eased in the following section.

1.2.3 Expanding the Entropy Model with History

In the previous section I introduced an entropy model with a history of length 0 (none). In this section I show how this entropy model can be expanded to use a history of the n chords.

A history of 1 is equivalent to examining the transitions between pairs of chords. Listing the transitions from the example introduced in Section 1.2 yields:

A′: C→C, C→C, C→C, C→F, F→F, F→F, F→F

B′: F→F, F→C, C→F, F→C, C→F, F→C, C→C

The number of possible events is now 4 instead of 2 (when only counting C and F chords as possible). Each event are expanded from a single chord into a transition.

Below is a table showing the distribution of events in piece A′ and B′:

Event	A′	B′
C→C	0.429	0.143
C→F	0.143	0.286
F→C	0.000	0.429
F→F	0.429	0.143

Piece B′ should have a higher entropy, since its distribution is closer to the equidistribution, which makes the piece harder to predict.

The exact entropy with history 1 can be computed as:

$$H(X) = -\sum_{i=2}^{|E|} P(E_{i-1}, E_i) \cdot log_2(P_{E_{i-1}}(E_i))$$

where $P(E_{i-1}, E_i)$ is the probability of seeing the pair of E_{i-1} and E_i while $P_{E_{i-1}}(E_i)$ is that of seeing E_i limited to events seen success E_{i-1}.

Using this I have computed the entropy of piece A and B to be:

Piece	History 0	History 1
A′:	1.0	0.46
B′:	1.0	0.86

B' has a much higher (almost doubled) entropy value than A' when using a history of 1. However, I believe that a history of 1 is still too small when it comes to musical pieces, since many typical chord transition are longer than this (see Williams, 2008).

1.2.4 Generalising History

In the previous section I discussed entropy values when using a history of 1. I now present a generalised mathematical model supporting n-length history.

First, I generalise the usage of the function P:

- $P(e_1, e_2, \ldots, e_n)$ is the probability that the events $e_1, e_2, \ldots, e_n$ occur sequentially.
- $P_{e_1,e_2,\ldots,e_{n-1}}(e_n)$ is the probability that events $e_1, e_2, \ldots, e_{n-1}$ is followed by e_n.

The probability of a single event is e_n is computed as the frequency:

$$P(e_n) := \frac{\mid \{e = e_n \mid e \in E\} \mid}{\mid E \mid}$$

The probability of seeing e_n after a number of other events, $e_1, e_2, \ldots, e_{n-1}$, is equal to the probability of seeing the preceding events times that of seeing e_n success these events:

$$P(e_1, e_2, \ldots, e_n) := P(e_1, e_2, \ldots, e_{n-1}) \cdot P_{e_1,e_2,\ldots,e_{n-1}}(e_n)$$

The function $P_{e_1,e_2,\ldots,e_{n-1}}(e_n)$ is therefore:

$$P_{e_1,e_2,\ldots,e_{n-1}}(e_n) = \frac{P(e_1, e_2, \ldots, e_n)}{P(e_1, e_2, \ldots, e_{n-1})}$$

Now that all functions are on a computational form, the entropy value with a history of n is computed as:

$$H(X) = -\sum_{e_1,e_2,\ldots,e_n \in E} P(e_1, e_2, \ldots, e_n) \cdot \log_2 \left(P_{e_1,e_2,\ldots,e_{n-1}}(e_n)\right)$$

This is equivalent to computing the probability of the history times the entropy of the events that follow the history:

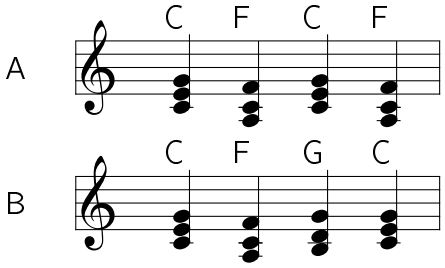
$$H(X) = -\sum_{e_1,e_2,\ldots,e_n \in E'} P(e_1, e_2, \ldots, e_n) \cdot H(\{e' \mid \{e_1, e_2, \ldots, e_n, e'\} \in X\})$$

where E' are the set of events that follow $e_1, e_2, \ldots, e_{n-1}$. This is how the entropy is computed in the application.

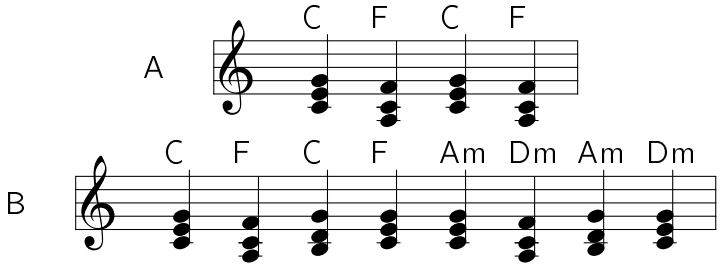
1.2.5 Phenomenons

Since the entropy is computed from an assumptions that all possible 36 variations chords (Section 1.2.1) could have been present in a work, a work with a high number of different chords will be likely to have a high entropy value.

An example of this is the following two pieces:



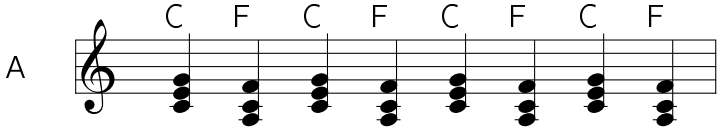
Even without regarding history, piece B obtain a higher entropy value (A is 1.0 while B is 1.5), since it uses more chords. This does however seem fair, since it does not repeat the chords as much as A.
Consider now the following two pieces:



The two pieces repeats the chords an equal amount of times (2 each). Piece B uses twice as many different chords than A, but is also twice as long.

The first 4 chords and the last 4 chords of B has the same entropy as A (1.0). But when joined together, the entropy raises to 2.0. This phenomenon has a rational explaination, since the entropy value is a measure of how many bits is needed to represent the state of each chord. 2 bits are in fact needed, to represent the choice from 4 different chords.

This phenomenon is also the reason why the following piece has the same entropy value as A:



Repeating the chords in A n times does not increase the entropy value.

1.2.6 Drawbacks

In this section I discuss the drawbacks of using entropy as a measure of predictability.

Concatinating two pieces with low entropy might yield one large piece with high entropy (see Section 1.2.5).

Theorem:
If piece A and B do not share any chords and $L_A > 0$, $L_B > 0$ are their respectable lengths:

$$H(A \mid B) > \frac{L_A}{L_A + L_B} H(A) + \frac{L_A}{L_A + L_B} H(B)$$

Proof:
Let the function F be defined as:

$$F(V, c) := -1 \cdot \sum_{e \in V} \cdot c \cdot P_e \cdot \log_2(c \cdot P_e)$$

since no chords are shared between A and B:

$$H(A \mid B) =$$

$$F(A, \frac{L_A}{L_A + L_B}) + F(B, \frac{L_B}{L_A + L_B})$$

Looking at the first part yields:

$$F(A, \frac{L_A}{L_A + L_B}) =$$

$$-1 \cdot \sum_{e \in A} \cdot \frac{L_A}{L_A + L_B} \cdot P_e \cdot \log_2(\frac{L_A}{L_A + L_B} \cdot P_e) =$$

$$-1 \cdot \sum_{e \in A} \cdot \frac{L_A}{L_A + L_B} \cdot P_e \cdot \Big(\log_2(P_e) + log_2(L_A) - log_2(L_A + L_B)\Big) > \Big(L_B > 0\Big)$$

$$-1 \cdot \frac{L_A}{L_A + L_B} \cdot \sum_{e \in A} \cdot P_e \cdot \log_2(P_e) =$$

$$\frac{L_A}{L_A + L_B} \cdot H(A)$$

Applying the same logic on the second part of the sum yields a lower bound for the entropy of the joined piece:

$$H(A \mid B) > \frac{L_A}{L_A + L_B} \cdot H(A) + \frac{L_B}{L_A + L_B} \cdot H(B)$$

When the A and B are of the same length, as the case of Section 1.2.5, the entropy value is bounded by:

$$H(A \mid B) > \frac{1}{2} \cdot (H(A) + H(B))$$

Which is the reason for the observed increased entropy.

1.3 Identifying Chords

In this section I go through the routine of identifying a chord from its notes. I achieve this using a recursive algorithm.

The main obstacle is to identify the root note. When the root note is known, reconstruction of the chord is achieved in a bottom-up manor.

If the root note, r, is known, the remaining notes of the chord can be found, by looking for a note, that is either one third, or one fifth higher than r. When a new chord note is found, this method is recursively repeated on the remaining notes, with the newly found note as root. Now, this assumed that the root note was already known. In order to find the correct root note, I try each note as the root note and pick the one that yields the longest chord. This procedure is described in the following pseudo-code:

```
fun identify(root, chord, notes):
  fifth = None
  for note in notes:
    if note is root+third:
       return chord+{note}+identify(root, chord, notes\{note})
    if note is root+fifths:
       fifths = note
  if fifths: # no third was found, use fifth
     return chord+fifth+identify(root, chord, notes\fifths)
  return chord
```

The `identify` function has a running time of $O(n)$. However since the root note is unknown, this function is called once per note, which leads to a total running time of $O(n^2)$. This quadratic running time is not a problem since n (the number of unique notes) has an upper bound of 24 in this context.

Chapter 2

Implementation Details

2.1 Overview

2.2 Input Data

2.3 Generation of MusicXML

2.4 SharpEye Pro

2.5 MusicXML Format

2.6 Improvement of Digitised Scores

2.7 Selected Database

2.1 Overview

In this section I present a brief overview of the implementation. For an in-depth description, see appendix C.

The following describes the steps needed to process music corpora from an external source: statistical results from external scores:

Step 1. Scores and metadata are collected from external sources

Step 2. The original format of the scores are JPEG. However the OCR engines only read monochrome bitmap files. The first step is therefore to

(a) convert the JPEG files into monochrome BMP files and

(b) apply the OCR software and store the result as MusicXML.

Step 3. The chords are identified from the notes.

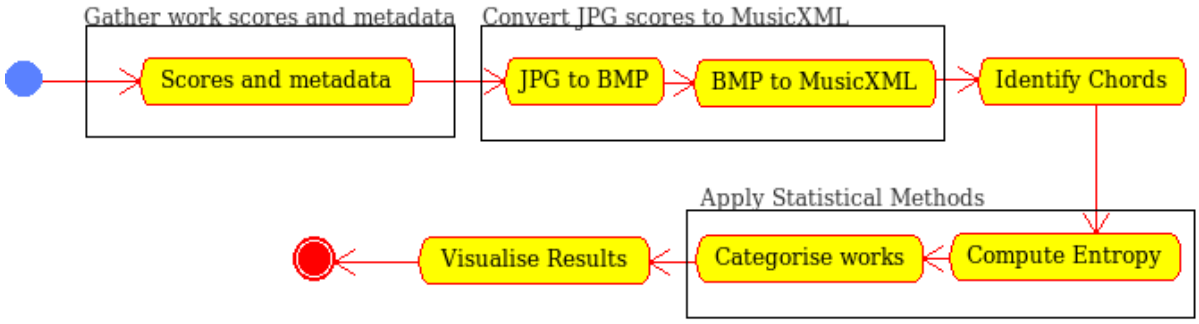
Step 4. The statistical results are computed. This is done by

(a) computing the entropy of each work and

(b) grouping the works by different criteria (e.g. category).

Step 5. The result can now be presented visually to ease the analysis task.

These steps are illustrated in the following diagram:



2.2 Input Data

In order to make any valuable statistical analysis I need to obtain access to music corpora.

Each piece of music should be in a practical format, not just for the convenience of this project, but for anyone who chooses to reuse the application code produced.

As of this writing, the most promising format for representing music is MusicXML. It is defined on top of XML, which makes its structure easy to parse using existing XML tools. Furthermore it is a popular format amongst notation software[1].

2.2.1 Possible Music Databases

I have searched for an appropriate public domain music database that could supply the musical data needed. In this section I briefly go through the databases found.

2.2.2 International Music Score Library Project

The International Music Score Library Project (IMSLP)[2] hosts more than 13.000 works, including Brahm's complete Piano pieces, Mozart's complete piano concerts and Corelli's complete works. The project is primarily driven by volunteers who submit scanned scores as PDF files. Because of this the quality of the scores vary greatly.

2.2.3 Mozarteum

Mozarteum[3] is a music project that focuses on Mozart only. Mozarteum is a free electronic version of the Neue Mozart-Ausgabe, which is a complete Mozart collection[4]. The online version is a reproduction of the original scores, created using notation software and then stored as JPEG files. Because the scores have been digitally recreated, they are of a much higher quality than those of IMSLP.

[1] fully supported by at least 25 music applications, http://www.recordare.com/xml/software.html

[2] www.IMSLP.org

[3] http://dme.mozarteum.at

[4] nma.at

2.2.4 Conclusion

Since none of the music databases hosts scores in the wanted MusicXML format, I have to extract this information from the formats available (JPEG and PDF).

None of the two mentioned music databases offers a packed version for download, nor do they support this by request[5]. Because of this I have to crawl the online web interfaces and collect the available metadata and scores. Afterwards the MusicXML representation must be generated from these files.

This process requires a lot more work than if the files had been available in MusicXML, but it is needed for the project to reach its goal of performing statistical analysis on music corpora.

[5] I tried e-mail requests.

2.3 Generating MusicXML from Scores

In this section, I describe how I generate MusicXML from scores obtained from an external music database. The scores can be in either BMP, TIFF or PDF format, however converting between these formats can be done using Image Magick[6].

2.3.1 Commercial Music OCR Software

In this section, I go over the commercial Music OCR (Optical Character Recognition) software I tested. All these applications have the feature of reading either a PDF, BMP or TIFF file, identifying the scores in the image and outputting this information as MusicXML. Writing such an OCR application is outside the scope of this project.
The OCR application should be able to:

- Generate MusicXML from a BMP, TIFF or PDF file
- Process multiple input files in an automated (non-interactive) manner

In the following table, I have summarized the features of each of the OCR applications I have found:

Name	PDF	BMP	TIFF	Automation
SharpEye Pro	yes	yes	yes	yes
SmartScore X Ultimate	yes	yes	yes	no
SmartScore Ultimate	yes	yes	yes	no
PDFtoMusic Pro	no	yes	yes	yes

SharpEye Pro

SharpEye Pro generates MusicXML files from images containing notes. The application is recommended by Recordare, the company behind MusicXML. It includes a command line version for processing image files in an automatic manner.

SmartScore X Ultimate

This application can convert PDF documents containing scanned notes to MusicXML. However, the application lacks an automation feature.

[6] www.imagemagick.org

PhotoScore Ultimate

This application can handle generation of MusicXML from PDF files, but does not include any automation. Also, it uses the same recognition engine as SharpEye Pro[7]

PDFtoMusic Pro

This application has an automation feature (batch mode), however it only works on PDF files created by notation software, since it relies on meta information written in the PDF documents. The files I have access to do not contain this meta information, which renders this application useless.

Choosing a Music OCR engine

The only application that could satisfy the two functionality demands was SharpEye Pro. It scans BMP images and has a command line version which enables easy automation.

However, SharpEye's command line utility is dysfunctional. A bug in the software renders it unable to output anything but SharpEye's own internal format named MRO. An update to resolve this problem is available, however the update contains another bug that makes the software unable to recognise a registration. Left is the choice between a piece of software that outputs poorly documented[8] MRO files or one that does not output at all.
Despite the bugs in SharpEye, I use this application to generate the needed MusicXML. In Section 2.4, I describe the process I went through in order to make it work.

2.3.2 Converting scores to BMP

Before I start generating the MusicXML information, I need to convert the input scores into a format that is recognised by the Music OCR software that will identify the actual notes.

In this process, I convert the input data into one common format. The format is chosen to be one non-compressed monochrome bitmap (BMP) file per page. The notation software has been optimised for scanned scores and monochrome BMP files with a width of 2750 pixels seems to meet there expectations of the software.

[7] SharpEye has been *integrated into [...] PhotoScore Ultimate*, http://www.visiv.co.uk/
[8] MRO documentation: http://www.visiv.co.uk/tech-mro.htm

2.4 SharpEye Pro

In this section, I describe how I made SharpEye Pro work in an automatic way, in order to solve the problems described in Section 2.3.1.
There are three possible ways of making SharpEye Pro usable:

1. Write a program that will convert MRO files into MusicXML
2. Write a program that controls SharpEye through the GUI to
   (a) read a MRO file and save it as MusicXML
   (b) read a BMP file and save the results as MusicXML

2.4.1 Convert MRO files into MusicXML

The first possibility I chose to test was that of writing a program that would parse the MRO files and output MusicXML. If this would work, I could still use the automation possibilities of the command line version.

The MRO file format

The MRO file format is designed to be easily parsed by a C program. Because of this all structures are in a format that resembles nested C-Arrays.
An example of this is the following "clef" structure from the MRO documentation:

```
"""
clef
{
shape Treble centre 29,34 pitchposn 2 }
"""
```

where "clef" is the name and "shape", "Treble" and "pitchposn" are its elements.

The structure of an MRO file is easy to parse, however the semantics of the values are not clearly documented. One example is the positioning system used. In the MRO documentation the units used for position coordinates are described by:

" All graphical coordinates increase to the right and down. They are written as row,column pairs, ie y,x. There are 16 units between stave lines in the

output, at least for the current version. Nearly all coordinates are in these units. Exceptions will be pointed out. "

The following MRO snippet, representing a chord, shows an example of how the positioning coordinates are used (from the documentation):

```
"""
chord
{
virtualstem False stemup True stemslash False
tuplettransform 1/1 tupletID -1 nofmmrestbars 0
accent False staccato False marcato False staccatissimo False tenuto False pause
naugdots 1 nflags 0 flagposn -31,166 headend -2 beam
{
id 1 nofnodes 1 nofleft 0 nofright 0 }
notes
{
nof 1
note
{
shape Minim staveoffset 0 p -2 accid None accid_dc 0 normalside True }

}
}
"""
```

According to the documentation, the "flagposn" element should note the position of the chord. But where exactly is the position of $y = -31$?

The documentation describes this element as: (r is y and c is x):

" [r,c] is the position of the flag or beam end of the stem on this chord. In the case of a stemless note (rest, breve, semi-breve) the c value is still valid, and is the centre of the note, chord or rest. "

This does not say anything about what point the coordinates are relatively computed from. When describing an unrelated chord, the documentation reads:

"[...] the flag position of this chord is [...] units from the left of the stave's top-left. "

Apparently, the coordinates should be relative to the stave's top left. However, this position does not match the graphical position of the note. Since I could not find any usable documentation on the semantics of the MRO format[9], I decided that writing a parser for this format would take too much time.

Controlling Windows Applications

Instead of parsing the MRO files, I decided to write a macro that would control the SharpEye Pro application by emulating a user. I use Visual Basic Scripting for all macro programming, since it provides easy access to the needed parts of the Windows API.

Below is an example of a Visual Basic Scripting (VBS) script:

```
Dim WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.run(''sharpeye.exe'')
WshShell.AppActivate(''SharpEye2'')
WshShell.SendKeys ''Hello SharpEye''
```

This script runs the SharpEye application, locates its main window and sends the keystrokes "Hello SharpEye" to it. The macros I use are able to open SharpEye and perform the follow actions:

1. Open an image file, run OCR recognition on it and save the result as MusicXML.

2. Open an MRO file and save it's content as MusicXML

[9] I requested such by email, but never received a reply

I tried both ways of automating the MusicXML convertion process and found the latter to be the most stable. Apparently it shows that controlling a Windows Application is an error prone task that might result it:

1. The wrong window being in focus (therefore receiving the input)
2. The window needed not being opened
3. The script being out-of-sync with the actual convertion process
4. One or more key-presses not being received by the targeted window

On top of these common faults, other errors might occur that are simpley not detected, since no error handling is possible. Because of this, I am using the simplest possible VBS script in the application.

However, it is possible to use the first approach and use it on other applications than just SharpEye Pro. In order to ease such a task, I have made a VBS script that, given an array of window titles and keys, finds the correct window and sends the keys to it, simplifying the task significantly. Using this script, the example above could be simplified to:

```
Keys = Array( _
        ''|SharpEye2/Registered to'', ''Hello SharpEye''
)
```

The script waits for a window with a title starting with either "SharpEye2" or "Registered to", sets it in focus and sends the keys "Hello SharpEye" to it. This generalised macro also supports unconditional pausing or pausing until a specific window to occur (like the pop-up that occurs when SharpEye has finished its recognition).

2.5 MusicXML

MusicXML is a popular music notation format by Recordare, that is supported by many notation applications. The goal of MusicXML is to provide an interchangeable format for score notation. The format allows two ways of ordering data inside the file:

1. *Partwise* is parts listing measures
2. *Timewise* is measures listing parts

The software used in this project handles files in the partwise format, however the other case can be handled using Recordare's XSL files for converting from one format to the other.

2.5.1 A Simple Example

Below is a small partwise MusicXML sample of a C note in C-major [10]:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 1.1 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="1.1">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

[10] http://www.recordare.com/xml/helloworld.html

The first `score-partwise` tag indicates that this MusicXML lists parts first. If this tag had been `score-timewise` the file would have been in the timewise format.

2.5.2 Parts

The `part-list` tag contains a list of parts in the score. Each part has an id together with various attributes listed as tags. Each part contains a list of measures as child tags.

2.5.3 Measures

Each measure can optionally contain attributes describing the key, the time unit etc. The `key` is the current key signature, represented by its number of flats or sharps. Positive counts are sharps, while negative counts are flats. In the example above the key is C-major, which has no flats nor sharps, so the `key` attribute is 0. Had the key been C-minor the `key` attribute would have been -3.

Since MusicXML is a notation format it also contains some purely notational information, such as the `clef` tag, that describes the position of the clef. Such notational tags are of no interest for this project and are therefore

As mentioned earlier measures can optionally contain the time unit. If no timing is given, the timing unit from the previous measure is used. The unit is defined by the `divisions` tag which states the number of divisions a quarter can be divided into. Each duration is then noted as a number of divisions. Each `note` tag has a `duration` stating the notes length in such divisions. Knowing the number of divisions the actual length, l, can be computed as:

$$l = \frac{1}{4} \cdot \frac{1}{\texttt{divisions}} \cdot \texttt{duration} = \frac{\texttt{durations}}{4 \cdot \texttt{divisions}}$$

2.5.4 Notes and Chords

The `note` tag contains a `pitch` tag describing the tone together with a `duration` tag describing the length in divisions. If the note is affected by either a flat or a sharp, the pitch tag will contain an `alter` tag describing the alternation (e.g. -1 for a flat), no matter if the alternation is local or global (part of the key signature or not).

Notes in MusicXML contains no starting time. Instead, the current time of an internal *clock* is used. The clock starts at zero and updates by the

duration for each note, allowing a new note to start where the previous note ended. The only exception of this is if the new note contains a `chord` tag. In this case, the clock must rewind to the beginning of the previous note, so the new note can be added at the same starting time.

To give the composer full control, MusicXML supports a `forward` and `backup` tag. Using these tags it is possible to change the internal clock by any amount in any direction. In case of a noted rest, MusicXML supports notes with an empty `rest` tag. This can be used where the rest is part of the notation and `forward` is inappropriate (its result being an *invisible* rest).

Because of the internal state, parsing MusicXML is not as simple as reusing an existing XML parser. The MusicXML parser will have to track of the internal clock at all times.

Notes can also represent percussion notation. In such cases, the note contains an `unpitched` tag, indicating that it represents a sound rather than a tone. Percussion notation is not interesting in the context of this project and such notes are therefore ignored.

2.5.5 Infelicities of MusicXML

There are some pitfalls of MusicXML. There exist cases that this format is not suited for. One obvious case is to extract non-continuous information. For example to search for all C-notes and return there position in the measure they were found in. This is a complicated task because all positions in MusicXML is relative to the previous note.

Below is the steps needed to complete the task:

1. Find all `notes` that contains a `pitch`, where `step` and `alter` combined gives a "C".
2. For each such note, scan backwards in the `measure` in order to compute the position in divisions.
3. Scan backwards until an `attribute` containing the `divisions` tag is found in a measure.
4. Compute the absolute position.

The problem here is the relative timings that MusicXML uses combined with its attempt to reduce redundancy by reusing previous attributes. A backward scan is only needed because the positional information is listed in an unknown measure.

Another downside of this design is that one misplaced note will shift the positions all previous notes in the measure.

The problem of positioning could have been solved by adding a `start` tag to each note. This tag would define where in the measure the current note would start.

But even with absolute positions a backward scan is still required to find the `divisions` number, that can be set anywhere in any measure.

This problem could be solved by choosing a simpler unit for durations. A straight forward unit would be to use fractions, just as the original sheets. This can be achieved by replacing the current duration with:

$$\texttt{duration}/(4 \cdot \texttt{divisions})$$

which will be the new fraction (e.g. $3/4$).

If MusicXML used both absolute positions and independent durations it would be easy to extract the position. However other information (e.g. key) is still located in some previous measure. This information should be explicitly stated in each measure to avoid misinterpretation.

According to Recordare, they chose the divisions model, because the MIDI format uses this[11]. However, they also claim MusicXML to be an interchangeable format, that prioritises clarity before compactness[12]. But absolute positions and durations, together with explicit attributes would have made this format much easier to interchange.

[11] http://recordare.com/xml/musicxml-tutorial.pdf
[12] www.recordare.com/xml/faq.htm

2.6 Improving the Quality of Digitised Scores

In this section, I present a simple method for improving the quality of digitised scores. The quality of such scores depends mainly on the original source and how this has been treated. Digitising a score by scanning a sheet of music is an error prone process that can result in added noise and gaps. I will try to improve the performance of popular OCR software when dealing with such scanned scores.
Below is an example of the first half of a measure from a scanned score[13]:

Many details are missing due to gaps, especially from the arms and notes. These parts are crucial for the OCR which can detect just 19 out of 24 notes in this measure. Also, SharpEye reports a total of 18 rhythm warnings on the page.

2.6.1 Extending segments vertically

A simple method to improve OCR performance is to close the gaps by stretching each pixel horizontally. This approach can be formalised as:

$$p'_{x,y} := p_{x,y} \wedge p_{x,y+1}$$

where $p_{x,y}$ is true iff the pixel at (x,y) is set (white) and $\wedge$ is the binary and operator. The method blackens all pixels that are either black or have a black pixel beneath it.
Running this method on the example from the previous section yields:

[13] Bach's "Wie schön leuchtet der Morgenstern", BWV 1

Although 1568 pixels have been flipped from white to black, there are still large gaps in the score and the improvement is small. The OCR software now recognises 18 notes in the measure, with a total of 17 rhythm warnings. A small improvement.

I now expand the procedure by another pixel, hence using:

$$p'_{x,y} := p_{x,y} \wedge p_{x,y+1} \wedge p_{x,y+2}$$

After flipping a total of 3025 pixels, the computation yields:

The notes are now more visible, however the double bars have merged into one. The OCR software recognises 18 notes, with a total of 23 rhythm warnings. This simple preprocessing does not seem to have any significant effect on the OCR software.

2.7 Database Selection

Based on the previous discussions of possible music databases and on my discussion on digitised scores, I have chosen to use Mozarteum as music database and will focus on this in the remainder of this paper.

The primary reason is the appealing quality of the scores hosted by Mozarteum. Despite my attempt to improve the quality of the user scanned scores at IMSLP, they are simply no match for the digitally generated scores at Mozarteum.

Although Mozarteum is listed as a complete collection of Mozart works, I was able to locate just 571 Mozart works out of a total of 626 possible. This collection should be large enough to conclude upon the works of Mozart, but choosing Mozarteum excludes any conclusions regarding music in general.

Chapter 3

Statistical Analysis

3.1 Methods

3.2 Results

3.3 Conclusion

3.1 Methods

In this section, I discuss the statistical methods I use to investigate trends in the chord progressions.

3.1.1 Interpolation

I use first degree polynomial interpolation, aka least square fitting (Lancaster and Salkauskas, 1986), to investigate whether the entropy in a region is raising or lowering.

I fit a first degree polynomial to a set of data points in the following way:

1. A first degree polynomial has the form $a \cdot x + b$, where a and b are constants.

2. The deviation between such a line and the data points in X is:

$$\sum_{x_i, y_i \in X} \big((a \cdot x_i + b) - y_i\big)^2$$

3. To find the best fit, I minimise the deviation. This is done by solving the following optimisation problem:

$$Minimise \sum_{x_i, y_i \in X} \big((a \cdot x_i + b) - y_i\big)^2$$

4. Such a quadratic convex optimisation problem can be solved using a Quadratic Programming library like `cvxopt` for Python or by implementing one of several methods for minimizing quadratic separable functions (e.g. Subgradient Method, (Heath, 2001)).

3.1.2 Correlation

Correlation is used to indicate the linear relationship between two variables.

To compute the correlation, I have used the Pearson product correlation coefficient. Let Z be a set of pairs x, y. The correlation between x and y can be computed as:

$$r = \frac{N \cdot [\sum_{x,y \in Z} x \cdot y] - [\sum_{x,y \in Z} x] \cdot [\sum_{x,y \in Z} y]}{[N \cdot \sum x^2 - (\sum x)^2][N \cdot \sum y^2 - (\sum y)^2]}$$

The correlation value is closely related to the linear fitting described in Section 3.1.1, since the correlation value is an indicator for the linearity in the data points.

I will assume a linear correlation when the correlation coefficient is higher than 0.3 or lower than -0.3 [1].

[1] These values have been suggested by Jakob Grue Simonsen, supervisor on the project

3.2 Results

In this section, I introduce and discuss the statistical results. As a start hypothesis, I assume that there is no correlation between entropy and any other property (e.g. year, length).

The categories used are equal to those of Mozarteum. In tables, the category names are shortened to ease readability (full names can be found in Appendix A).

When discussing correlation coefficients, I use the terms *weak* and *strong* as a measure of the coefficient's absolute distance from zero. Coefficients below 0.3 are *weak*, while those of 0.3 or higher are *strong*.

I denote the *mass* of a group as the amount of works in the group (e.g. a specific category).

3.2.1 Ordered by Year

In this section, I investigate possible correlations between production year and entropy. Specifically, I test the hypotheses:

Hypothesis A: *There is no relationship between production year and entropy, i.e. the correlation of these values are below* 0.3.

Hypothesis B: *The relationship between production year and entropy values are constant over time, i.e. deviation of such is below* 10 *percent.*

Without History

The Figure below suggests no falsification of Hypothesis A, since no correlation between entropy values and production year seems to exist. In fact, the entropy values are so high, they seem almost constant.

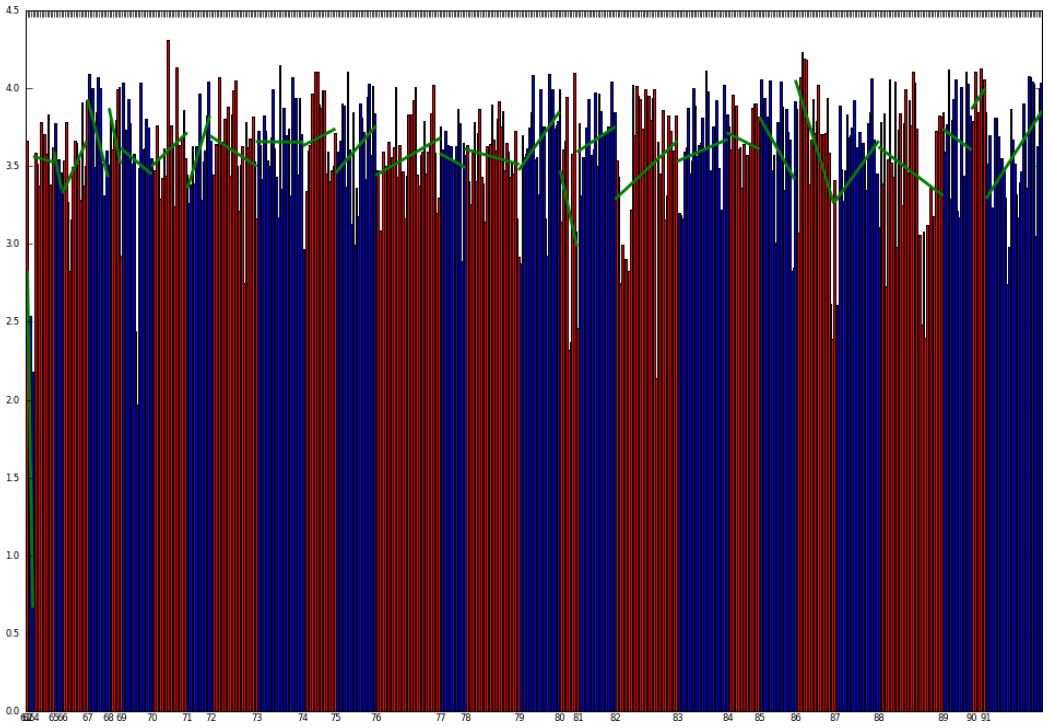


Figure 3.1: Entropy value ordered by year

The correlation coefficient is 0.061, which is too weak to falsify the hypothesis. It seems the production year does not influence the complexity when disregarding history.

To test Hypothesis B I split Mozart's life into three time periods:

1. **childhood:**
   Mozart's first pieces, including all works he composed before turning 17 years of age.

2. **early adulthood:**
   The period 1773 - 1781 where he worked under various employees, however mainly the Salzburg Court.

3. **late adulthood:**
   The period 1782 to 1791 which stretches from Mozart's move to freelance work only and to his death in 1791.

I now list the correlation between production year and entropy in each of the three periods:

Period	Mass	Correlation
1756 - 1772	137	0.30
1773 - 1781	231	-0.080
1782 - 1791	267	0.045

The correlation coefficient in works produced in Mozart's early years is much higher (0.255) than those of later works. The deviation is 0.025 (28.3 percent), falsifying Hypothesis B.

History of 1

In this section, I investigate Hypothesis A and B using a history of 1.

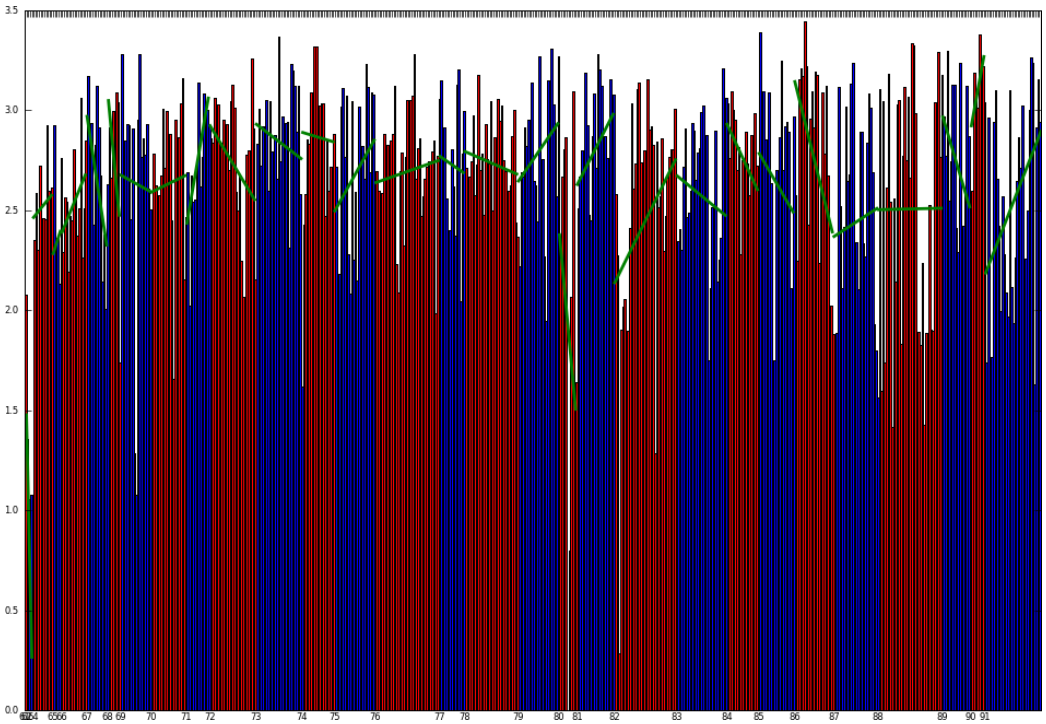


Figure 3.2: Entropy value ordered by year

The correlation coefficient is now 0.011, which is too weak to falsify Hypothesis A. As in Section 3.2.1, I test Hypothesis B by listing the correlation in different periods:

Period	Mass	Correlation
1756 - 1772	137	0.42
1773 - 1781	231	-0.14
1782 - 1791	267	0.052

Once again, a significant correlation shows in the early years, but not in the later. All the coefficients has grown stronger, especially in the *early adulthoot* (75 percent), however the only significant correlation is in to be seen in the *childhood*. Also, the correlation in this period is still by far the strongest. The deviation is now 0.054 (48.8 percent), hence falsifying Hypothesis B.

History of 2

In this section, I investigate Hypothesis A and B using a history of 2.

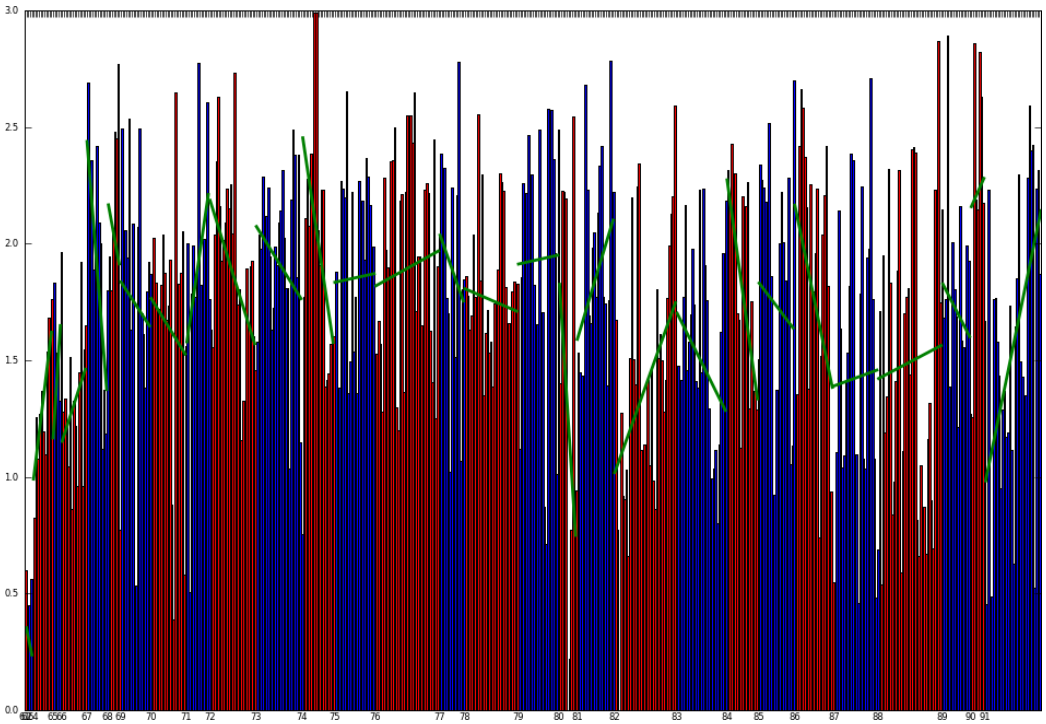


Figure 3.3: Entropy value ordered by year

The correlation between production year and entropy values is now -0.038. Once again, much too weak to falsify Hypothesis A.

As in the previous sections, I list the correlation in different time periods:

Period	Mass	Correlation
1756 - 1772	137	0.46
1773 - 1781	231	-0.13
1782 - 1791	267	0.084

Once again, the *childhood* period is the only significant correlation and is still by far the strongest. The deviation is now 0.059 (43.1 percent); once again, falsifying Hypothesis B.

3.2.2 Ordered by Category

In this section I investigate possible correlations between work category and entropy. Specially, I test the following two hypothesis:

Hypothesis C: *There is no relationship between a work's category and it's entropy, i.e. the averages of entropy values in categories do not vary more than* 1.0.

Hypothesis D: *The correlation between production year and entropy is independent of category, i.e. the vary of such is below* 10 *percent.*

Without History

In this section, I investigate Hypothesis C and D without using any history.

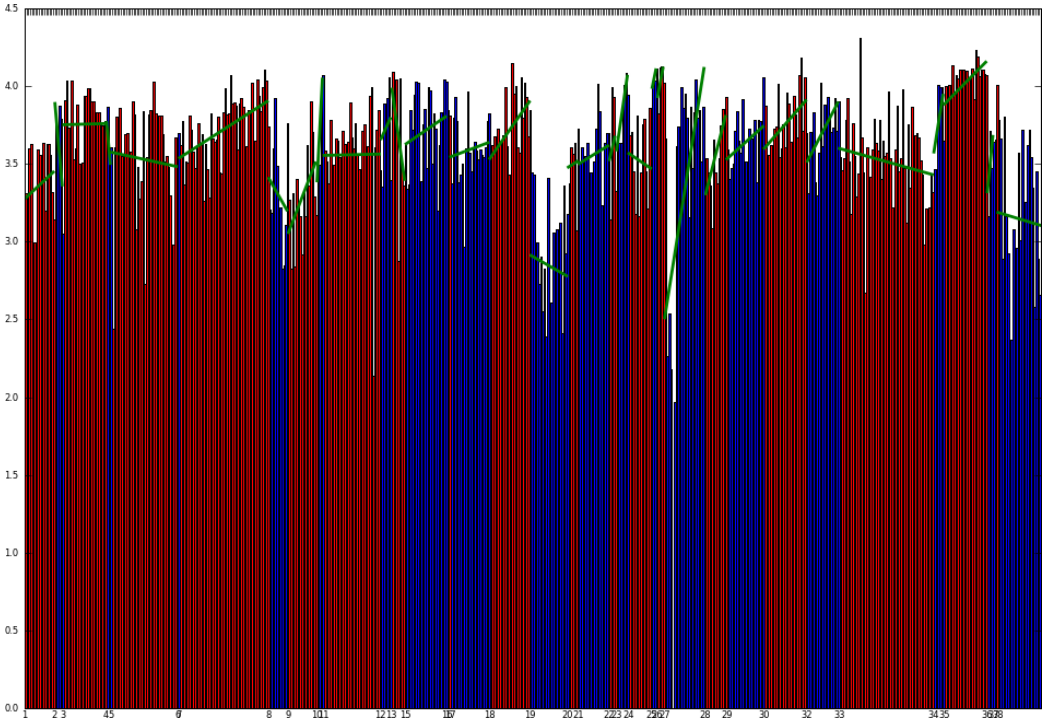


Figure 3.4: Entropy value ordered by category

Again, the entropy values are high when not using history. There is no clear global correlation, however there seems to be hints of correlation within specific categories.

To test Hypothesis C, I compute the average entropy for each category:

Index	Category	Mass	Period	Average
0	arias, scenes, v...	53	1765 - 1791	3.52
3	canons	21	1772 - 1788	**2.85**
4	cassations, sere...	24	1766 - 1787	3.76
5	concertos for on...	24	1773 - 1791	3.75
6	concertos for on...	23	1774 - 1791	3.72
7	dances	37	1769 - 1791	3.53
8	divertimentos an...	12	1773 - 1782	3.56
12	marches	13	1769 - 1788	3.52
13	masses	22	1766 - 1782	3.59
15	operas	24	1767 - 1791	**4.02**
18	part songs	11	1765 - 1788	3.31
19	piano pieces	23	1761 - 1791	3.31
20	piano sonatas	21	1774 - 1789	3.64
21	piano trios	18	1764 - 1791	3.70
25	smaller pieces o...	17	1766 - 1791	3.56
26	sonatas and vari...	32	1764 - 1788	3.56
27	sonatas for orga...	17	1767 - 1780	3.36
28	songs	25	1772 - 1791	3.15
29	string quartets	23	1770 - 1790	3.72
33	symphonies	50	1764 - 1788	3.72
34	variations for p...	16	1766 - 1791	3.28

The highest value is 4.02 (operas), while the lowest is 2.85 (canons). The average is 3.53 and the deviation is 0.25, which is 13.4 percent. The deviation is just over 10 percent which falsifies Hypothesis C.

To investigate Hypothesis D, I list the correlation between year and entropy in each category[2]:

Index[3]	Category	Mass	Period	Correlation
0	arias, scenes, v...	53	1765 - 1791	-0.19
3	canons	21	1772 - 1788	-0.29
4	cassations, sere...	24	1766 - 1787	-0.019
5	**concertos for on...**	24	1773 - 1791	**0.42**
6	concertos for on...	23	1774 - 1791	0.25
7	dances	37	1769 - 1791	-0.032
8	**divertimentos an...**	12	1773 - 1782	**0.72**
12	marches	13	1769 - 1788	-0.22
13	masses	22	1766 - 1782	0.10
15	**operas**	24	1767 - 1791	**0.34**
18	part songs	11	1765 - 1788	0.069
19	**piano pieces**	23	1761 - 1791	**0.67**
20	**piano sonatas**	21	1774 - 1789	**0.35**
21	**piano trios**	18	1764 - 1791	**0.68**
25	smaller pieces o...	17	1766 - 1791	0.23
26	sonatas and vari...	32	1764 - 1788	0.081
27	sonatas for orga...	17	1767 - 1780	0.065
28	songs	25	1772 - 1791	-0.068
29	**string quartets**	23	1770 - 1790	**0.46**
33	**symphonies**	50	1764 - 1788	**0.46**
34	**variations for p...**	16	1766 - 1791	**0.47**

Several categories has a correlation coefficient above 0.30 (marked with bold). Especially *divertimentos and serenades for wind instruments* (number 8). The average is 0.22 and the deviation 0.29 (135.7 percent). This implies that the correlation is not constant over categories, which falsifies Hypothesis D. A more interesting observation is perhaps, that categories that has a positive correlation, dominates the list (15 against 6). Also, all of the significant correlations are positive.

[2]Only categories with at least 10 works is included

History of 1

In this section, I investigate Hypothesis C and D using a history of 1.

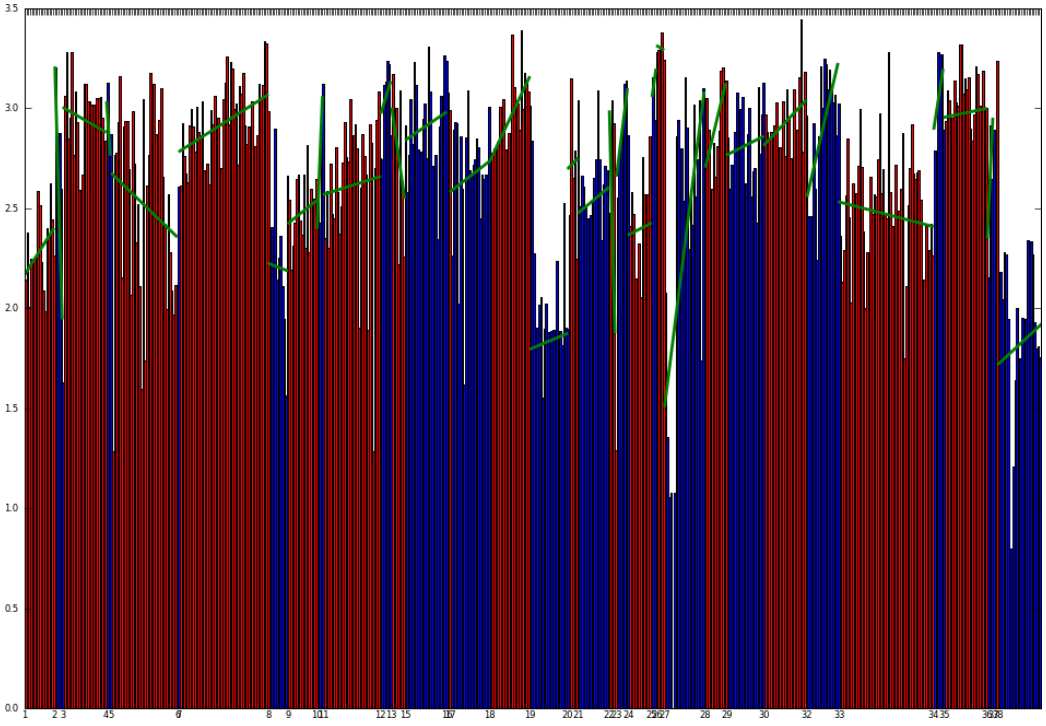


Figure 3.5: Entropy value ordered by category

The entropy values seem more distinct, than in the previous section.

To test Hypothesis C, I compute the average entropy for each category:

Index	Category	Mass	Period	Average
0	arias, scenes, v...	53	1765 - 1791	2.47
3	canons	21	1772 - 1788	1.84
4	cassations, sere...	24	1766 - 1787	2.94
5	concertos for on...	24	1773 - 1791	2.93
6	concertos for on...	23	1774 - 1791	2.91
7	dances	37	1769 - 1791	2.52
8	divertimentos an...	12	1773 - 1782	2.91
12	marches	13	1769 - 1788	2.40
13	masses	22	1766 - 1782	2.66
15	operas	24	1767 - 1791	**2.98**
18	part songs	11	1765 - 1788	2.21
19	piano pieces	23	1761 - 1791	2.29
20	piano sonatas	21	1774 - 1789	2.81
21	piano trios	18	1764 - 1791	2.89
25	smaller pieces o...	17	1766 - 1791	2.54
26	sonatas and vari...	32	1764 - 1788	2.61
27	sonatas for orga...	17	1767 - 1780	2.28
28	songs	25	1772 - 1791	**1.82**
29	string quartets	23	1770 - 1790	2.95
33	symphonies	50	1764 - 1788	2.93
34	variations for p...	16	1766 - 1791	2.48

The average is now 2.59 and the deviation 13.5 percent, which falsifies Hypothesis C. An interesting observation is that, as with no history, *operas* score highest. *Songs* are now the lowest, closely followed by *canons*.

To investigate Hypothesis D, I list correlation between year and entropy for each category[4] when using a history of 1:

Index	Category	Mass	Period	Correlation
0	arias, scenes, v...	53	1765 - 1791	-0.098
3	canons	21	1772 - 1788	-0.22
4	cassations, sere...	24	1766 - 1787	-0.23
5	**concertos for on...**	24	1773 - 1791	**0.31**
6	concertos for on...	23	1774 - 1791	0.25
7	dances	37	1769 - 1791	-0.15
8	**divertimentos an...**	12	1773 - 1782	**0.65**
12	marches	13	1769 - 1788	-0.05
13	masses	22	1766 - 1782	0.12
15	operas	24	1767 - 1791	0.044
18	**part songs**	11	1765 - 1788	**0.44**
19	**piano pieces**	23	1761 - 1791	**0.70**
20	piano sonatas	21	1774 - 1789	0.12
21	**piano trios**	18	1764 - 1791	**0.84**
25	smaller pieces o...	17	1766 - 1791	-0.032
26	sonatas and vari...	32	1764 - 1788	0.17
27	**sonatas for orga...**	17	1767 - 1780	**0.42**
28	songs	25	1772 - 1791	0.11
29	**string quartets**	23	1770 - 1790	**0.57**
33	**symphonies**	50	1764 - 1788	**0.38**
34	variations for p...	16	1766 - 1791	0.29

An interesting observation, is that *divertimentos and serenades for wind instruments* (number 8), which showed the highest correlation without history (Section 3.2.2) has now dropped to 0.65. On the other hand, *piano pieces* and *piano trios* has raised.

The average is now 0.22 and the deviation 0.30 (135.8 percent), which again falsifies Hypothesis D. All significant correlation coefficients are still suggesting positive relationships and the total number of positive coefficients still outnumber that of negative coefficients (15 against 6).

[4]Only categories with at least 10 works is included

History of 2

In this section, I investigate Hypothesis C and D using a history of 2.

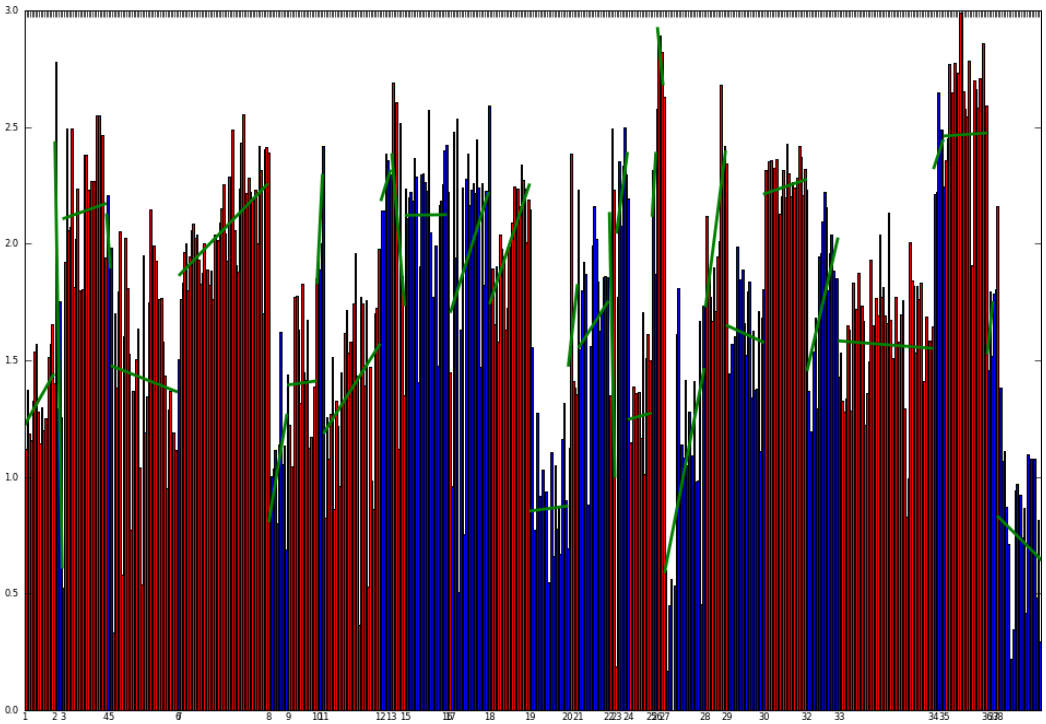


Figure 3.6: Entropy value ordered by category

Once again, raising the history length seems to raise the contrast as well.

To test Hypothesis C, I compute the average entropy for each category:

Index	Category	Mass	Period	Average
0	arias, scenes, v...	53	1765 - 1791	1.57
3	canons	21	1772 - 1788	0.86
4	cassations, sere...	24	1766 - 1787	2.14
5	concertos for on...	24	1773 - 1791	2.24
6	concertos for on...	23	1774 - 1791	2.12
7	dances	37	1769 - 1791	1.42
8	divertimentos an...	12	1773 - 1782	2.06
12	marches	13	1769 - 1788	1.26
13	masses	22	1766 - 1782	1.96
15	operas	24	1767 - 1791	**2.47**
18	part songs	11	1765 - 1788	1.04
19	piano pieces	23	1761 - 1791	1.03
20	piano sonatas	21	1774 - 1789	1.61
21	piano trios	18	1764 - 1791	1.74
25	smaller pieces o...	17	1766 - 1791	1.65
26	sonatas and vari...	32	1764 - 1788	1.38
27	sonatas for orga...	17	1767 - 1780	1.33
28	songs	25	1772 - 1791	**0.74**
29	string quartets	23	1770 - 1790	2.00
33	symphonies	50	1764 - 1788	2.06
34	variations for p...	16	1766 - 1791	1.40

The average is 1.62 and the deviation 0.47 (29.1 percent), which falsifies Hypothesis C, once more. *Operas* still score highest, while *songs* score lowest.

To test Hypothesis D, I list the correlation between year and entropy for each category[5] when using a history of 2:

Index	Category	Mass	Period	Correlation
0	arias, scenes, v...	53	1765 - 1791	-0.0076
3	canons	21	1772 - 1788	-0.27
4	cassations, sere...	24	1766 - 1787	-0.017
5	concertos for on...	24	1773 - 1791	-0.039
6	concertos for on...	23	1774 - 1791	0.10
7	dances	37	1769 - 1791	-0.026
8	**divertimentos an...**	12	1773 - 1782	**0.77**
12	marches	13	1769 - 1788	-0.066
13	masses	22	1766 - 1782	0.27
15	operas	24	1767 - 1791	-0.011
18	**part songs**	11	1765 - 1788	**0.61**
19	**piano pieces**	23	1761 - 1791	**0.58**
20	piano sonatas	21	1774 - 1789	-0.20
21	**piano trios**	18	1764 - 1791	**0.71**
25	smaller pieces o...	17	1766 - 1791	-0.024
26	**sonatas and vari...**	32	1764 - 1788	**0.36**
27	**sonatas for orga...**	17	1767 - 1780	**0.41**
28	songs	25	1772 - 1791	-0.19
29	**string quartets**	23	1770 - 1790	**0.71**
33	**symphonies**	50	1764 - 1788	**0.44**
34	variations for p...	16	1766 - 1791	0.16

Several of the correlations are higher than the significant 0.3, including *piano pieces*, *sonates* and *symphonies*. The average is 0.20 and the deviation 0.32 (159.0 percent). This is far higher than the 0.10 percent threshold, which falsifies Hypothesis D. Once more, all significant coefficients are positive, however the total number of positive relationships has lowered from 15 to 11, and thus balancing the score (11 against 10). Many of the negative coefficients are however very weak. Ignoring coefficients weaker than 0.05 yields a score of 11 positive against 4 negative. As can be seen, all the positive coefficients is stronger than 0.05.

[5]Only categories with at least 10 works is included

3.2.3 Ordered by Length

In this section I investigate possible correlations between the entropy and the number of chords used in a work. Specifically, I test the following hypotheses:

Hypothesis E: *There is no relationship between the length of a work and it's entropy value, i.e. the correlation of such is below* 0.3.

Hypothesis F: *There is no relationship between the category of a work and it's length, i.e. the average work length in the categories do not vary more than* 10 *percent.*

Hypothesis G: *The correlation between production year and work length is constant, i.e. the correlation between such is does not vary between category with more than* 0.15.

Without History

In this section, I investigate Hypothesis E, F and G without the use of history.

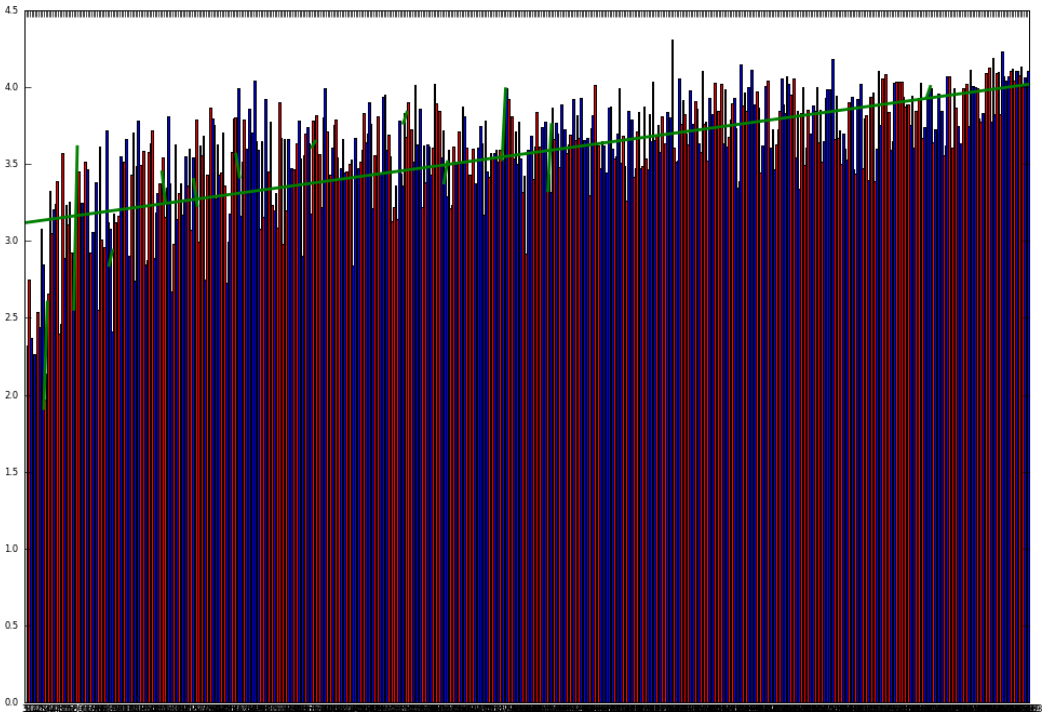


Figure 3.7: Entropy value ordered by the number of chords

The correlation coefficient is 0.38, which is a bit over the significance threshold, which falsifies Hypothesis E.

To test Hypothesis F, I compute the average length for each category:

Index	Category	Mass	Period	Chords
0	arias, scenes, v...	53	1765 - 1791	749
3	canons	21	1772 - 1788	110
4	cassations, sere...	24	1766 - 1787	2406
5	concertos for on...	24	1773 - 1791	3330
6	concertos for on...	23	1774 - 1791	2351
7	dances	37	1769 - 1791	549
8	divertimentos an...	12	1773 - 1782	1609
12	marches	13	1769 - 1788	324
13	masses	22	1766 - 1782	3118
15	operas	24	1767 - 1791	**14269**
18	part songs	11	1765 - 1788	199
19	piano pieces	23	1761 - 1791	300
20	piano sonatas	21	1774 - 1789	667
21	piano trios	18	1764 - 1791	870
25	smaller pieces o...	17	1766 - 1791	948
26	sonatas and vari...	32	1764 - 1788	445
27	sonatas for orga...	17	1767 - 1780	343
28	songs	25	1772 - 1791	**101**
29	string quartets	23	1770 - 1790	1457
33	symphonies	50	1764 - 1788	1813
34	variations for p...	16	1766 - 1791	364

An interesting observation here, is that *operas*, that had the highest average entropy, also has the highest average length, while *songs*, with the lowest average entropy, has the lowest average length.

The average is 1729.62 while the deviation is 2964.67 (41.7 percent), which falsifies Hypothesis F.

Since this computation is independent of entropy, it will not change with the history length.

To test Hypothesis G, I compute the correlation between production year and work length:

Index	Category	Mass	Period	Correlation
0	arias, scenes, v...	53	1765 - 1791	0.065
3	canons	21	1772 - 1788	**-0.62**
4	cassations, sere...	24	1766 - 1787	0.19
5	concertos for on...	24	1773 - 1791	-0.55
6	concertos for on...	23	1774 - 1791	0.036
7	dances	37	1769 - 1791	-0.026
8	divertimentos an...	12	1773 - 1782	0.76
12	marches	13	1769 - 1788	0.16
13	masses	22	1766 - 1782	0.16
15	operas	24	1767 - 1791	0.11
18	part songs	11	1765 - 1788	0.33
19	piano pieces	23	1761 - 1791	0.46
20	piano sonatas	21	1774 - 1789	0.030
21	piano trios	18	1764 - 1791	0.70
25	smaller pieces o...	17	1766 - 1791	-0.093
26	sonatas and vari...	32	1764 - 1788	0.47
27	sonatas for orga...	17	1767 - 1780	0.53
28	songs	25	1772 - 1791	-0.32
29	string quartets	23	1770 - 1790	**0.79**
33	symphonies	50	1764 - 1788	0.58
34	variations for p...	16	1766 - 1791	0.29

The average is 0.21 and the deviation 0.35 (167.2 percent). This deviation falsifies Hypothesis G, since the correlation do change over category.

Since this computation is independent of entropy, it will not change with the history length.

History of 1

In this section, I investigate Hypothesis E, F and G using of a history of 1.

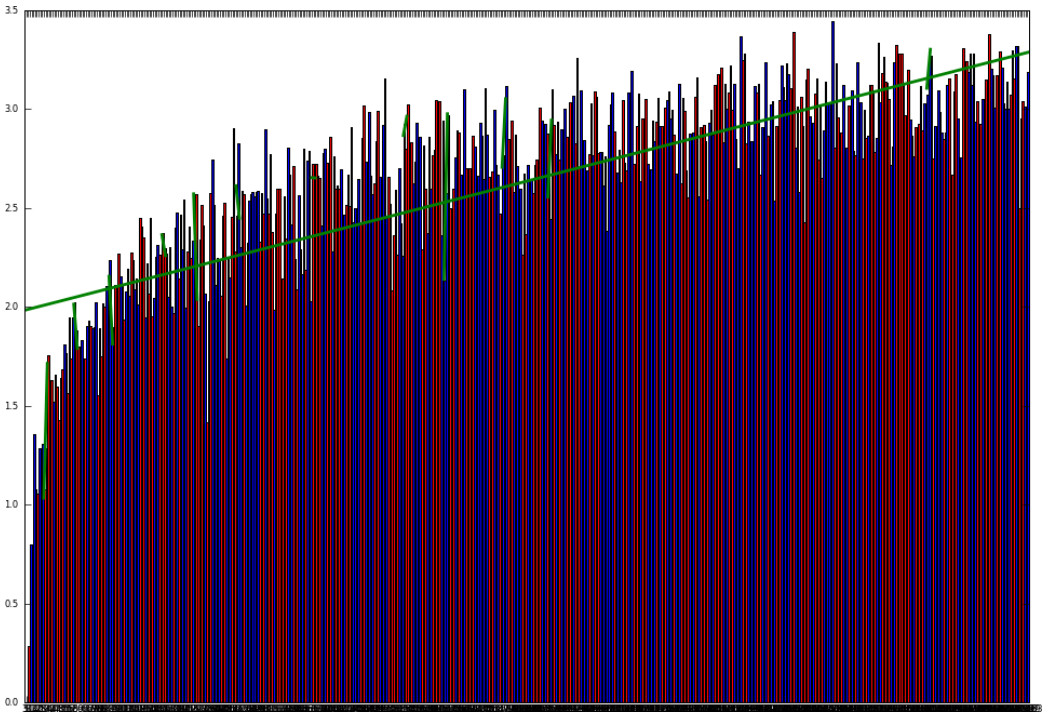


Figure 3.8: Entropy value ordered by the number of chords

The correlation coefficient is now 0.36, which falsifies Hypothesis E. It looks like the relationship might however be stronger in early works than in later. Computing the correlation coefficient of works in the time periods used in Section 3.2.1 yields:

Period	Mass	Correlation
1756 - 1772	137	0.32
1773 - 1781	231	0.44
1782 - 1791	267	0.35

Surprisingly, this shows a strong relationship in the middle years, and not the beginning as would have been expected from reading the graph.

History of 2

In this section, I investigate Hypothesis E, F and G using of a history of 2.

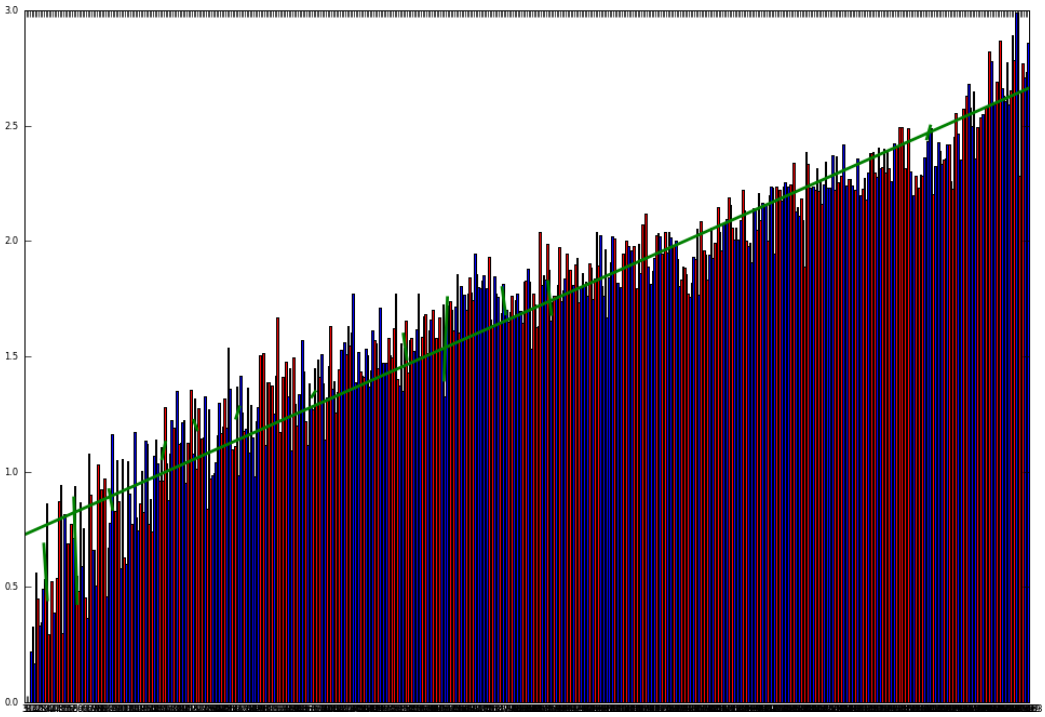


Figure 3.9: Entropy value ordered by the number of chords

The correlation coefficient between lengths and entropy values are now 0.59. This is higher than the 0.44 coefficient seen in the previous section and suggests that longer works do have higher entropy values.

3.2.4 Conclusion

In the previous sections I have falsified the following Hypotheses:

Hypothesis B: *The relationship between production year and entropy values are constant over time, i.e. deviation of such is below* 10 *percent.*

Hypothesis C: *There is no relationship between a work's category and it's entropy, i.e. the averages of entropy values in categories do not vary more than* 1.0.

Hypothesis D: *The correlation between production year and entropy is independent of category, i.e. the vary of such is below* 10 *percent.*

Hypothesis E: *There is no relationship between the length of a work and it's entropy value, i.e. the correlation of such is below* 0.3.

Hypothesis F: *There is no relationship between the category of a work and it's length, i.e. the average work length in the categories do not vary more than* 10 *percent.*

Hypothesis G: *The correlation between production year and work length is independent of category, i.e. the correlation between such does not vary between category with more than* 10 *percent.*

While Hypothesis A was not falsified, i.e. it might be true that:

Hypothesis A: *There is no relationship between production year and entropy, i.e. the correlation of these values is below* 0.3.

Production Year and Entropy

I conclude that there is a positive relationship between production year and entropy values in the following five categories:

- Divertimentos and serenades for wind instruments
- Piano pieces
- Piano trios
- String quartets
- Symphonies

I base this on the observation, that all five categories had a significant positive correlation coefficient using all three history settings.

I also conclude, that this relationship was significantly stronger in the first 16 years of Mozart's life (with a correlation of 0.46).

Category and Entropy

With regard to categories, I conclude that operas generally has higher entropy than other works, and that songs and canons have lower. I base this on the observation, that operas scored highest in average during all history settings, while songs and canons scored lowest.

When using a history of 2 operas are 0.85 higher than the average, which is a deviation of 81 percent more than the standard. Likewise, songs deviated with 87 percent more than the standard and canons with 61 percent more.

Work Length and Entropy

I conclude that longer works has higher entropy. I base this on the observation of the strong correlation of 0.59 with the history length 2.

I have not been able to falsify that the other observed relationships are caused by the lengths of the works, rather than the entropy, however this does seem likely at the moment, since the correlation between lengths and entropy is strong at all history settings.

3.3 Conclusion

I show that the entropy values of Mozart works grow year by year in at least 5 categories, specifically *divertimentos and serenades*, *piano pieces*, *piano trios*, *string quartets* and *symphonies*. I show that this is also the case across categories during Mozart's first 16 years (with a correlation of 0.46). I conclude that operas are most likely to have a high entropy value (2.47 in average), whereas finding such in canons and songs are least likely (0.86 and 0.74 in average respectively). I also show that, in the case of Mozart, longer works have higher entropy values (with a correlation of 0.59).

All of these results are based on entropy as a measure of predictability. However, as described in Section 1.2.6, this does have it's drawbacks. The result that longer works have higher entropy values might be due to the drawbacks of using entropy as a measure, or it might be because Mozart actually did variate longer works more. I have not been able to verify nor falsify if the other observations are due to the length of the works, so they might be results of drawbacks in the entropy model. However, I did observe, that the correlation between Mozart's early works and their lengths did not match with that of their entropy.

The framework I have implemented is modular by design, thereby allowing easy manipulation and expansion. It currently implements the functions needed for working with MusicXML and does this in such a generic manner, that extracting more features of MusicXML could be implemented. Furthermore, it implements an API for extracting MusicXML information from notes. The current framework only implements the entropy method for measuring complexity, however it is prepared for the addition of new methods.

It would be interesting to see more analysis on the use of entropy as a measure of complexity, specifically on how it increases by the length and how this affects the results. Even more interesting perhaps, would be to see other statistical methods applied on the collected data, e.g. Markov Chains (Norris, 1998). More scores could be collected, including other artists. This would open the possible of a more general analysis, or perhaps a comparison of different artists. Such later projects could also extract more information about the pieces and thereby analysis the melody structures, intensity, modulations etc. All these features could be implemented in the provided framework, thus allowing multiple collection and analysis schemes for all parties involved.

Bibliography

Michael T. Heath. Scientific Computing: An Introductory Survey. McGraw-Hill Higher Education, 2. edition, 2001.

James L. Johnson. Probability and Statistics for Computer Science. Wiley-Interscience, 2008.

Peter Lancaster and Kestutis Salkauskas. Curve and Surface Fitting: An Introduction. London: Academic Press, 1986.

Elizabeth H. Margulis. A model of melodic expectation. Music Perception, 22(4):663–714, 2005.

Elizabeth H. Margulis and Andrew P. Beatty. Musical style, psychoaesthetics, and prospects for entropy as an analytic tool. Music Perception, 32(4), Winter 2008.

James R. Norris. Markov Chains (Cambridge Series in Statistical and Probabilistic Mathematics). Cambridge University Press, new edition, 1998.

Erik Ray. Learning XML. O'Reilly Media, 2. edition, 2003.

Claude E. Shannon. A mathematical theory of communication. Bell Systems Technical Journal, 27:379–423, 1948.

Andrew Surmani. Essentials of Music Theory, Book 1. Alfred Publishing Company, 1998.

David Temperley. Music and Probability. MIT Press, 2007.

Jessica Williams, editor. Easy 4-Chord Keyboard Songbook: Popular Hits. Music Sales, 2008.

Appendix A

Work Categories

The works processed in this project was partioned in the following categories[1]:

1. arias, scenes, vocal ensembles and choirs with orchestra arrangements of works by George Frederic Handel
2. canons
3. cassations, serenades, and divertimentos for orchestra
4. concertos for one or more pianos and orchestra
5. concertos for one or more string, wind or plucking instrument and orchestra
6. dances
7. divertimentos and serenades for wind instruments
8. divertimentos for string and wind instruments
9. duos and trios for strings and wind instruments
10. litanies, vespers
11. marches
12. masses
13. music for pantomimes and ballets

[1]The same categories as can be found on www.mozarteum.com

14. operas
15. oratorios, sacred singspiele, and cantatas
16. part songs
17. piano pieces
18. piano sonatas
19. piano trios
20. quartets with one wind instrument
21. quintets with wind instruments
22. smaller pieces of church music
23. sonatas and variations for piano and violin
24. sonatas for organ and orchestra
25. songs
26. string quartets
27. string quintets
28. symphonies
29. variations for piano
30. works for four handed piano
31. works for two pianos
32. works of doubtful authenticity

Appendix B

Missing Works

This is a list of Köchel Numbers, that was not part of the data used in this paper (89 in total):

17	18	44	46
53	54	55	56
57	58	59	60
61	64	91	92
93	98	102	107
115	116	139	140
142	149	150	151
153	154	163	177
178	182	184	187
190	197	198	199
200	205	206	221
226	227	235	236
267	268	287	288
290	291	312	324
325	326	327	340
342	350	355	362
372	382	389	393
395	405	410	411
414	429	434	443
444	470	510	514
560	565	569	577
579	611	615	624
625			

Appendix C

Implementation Details

To speed up the computations, I have distributed the step involving conversion from BMP files to MusicXML (the bottleneck). This is done by running multiple workers, each in a seperate virtual machine. Doing so eases distribution over cores as well as physical machines.

The process starts by downloading scores from Mozarteum and ends with a visualisation of the statistical results.

C.0.1 Ripping http://dme.mozarteum.net

The ripping of Mozarteum consists of downloading each JPEG page available on the site.

The pages are located at the URL:
http://dme.mozarteum.at/DME/nma/scan.php?vsep=*VSEP*&p1=*P1*&l=*L*
Where *VSEP* is a collection number, *P1* is a page in the collection and *L* is the language (2 is English). The viewer will display pages 10 at a time, starting from page *P1*.

The collection has nothing to do with the Mozart work the page is part of. The page might not be part of a score at all.

The ripper disregards this and downloads all pages for all collections and stores them as COLLECTION/PAGE.jpg.

The crawler will order the pages according to the work.

rip_mozart.py

The script works by using a simple regular expression that will find all page listings in the 10-page viewer at:

http://dme.mozarteum.at/DME/nma/scan.php?vsep=*VSEP*&p1=*P1*&l=*L*

The PHP file displays pages 10 at a time. A complete work is downloaded using the following algorithm:

1. Let P be the first page number
2. Download all pages listed from P (max 10)
3. If 10 pages were downloaded, add 10 to P and repeat 2)
4. Stop when less than 10 pages are listed from P

Each page is extracted from the HTML using a simple regular expressing that identifies the "src" attribute of the "<img>" tags.

The ripper uses the third-party URLGrabber module to simplify grabbing with retry. Keep-Alive does (strangely enough) not work for Mozarteum, resulting in a new connection on each request without the old one being closed. Because of this, Keep-Alive has been disabled.

C.0.2 Crawling http://dme.mozarteum.net

The Crawler locates each work on Mozarteum by searching for each available category. This eases the task of ordering by category.

The script extracts the VSEP collection number and the start page from the score link. It then fetches the pages needed from the data storage, that was previously populated by the Ripper. In order to know when the current work ends (on which page), it checks the start page of the succeeding work. If no succeeding work exists, it will assume that the work is the last in the collection and use all remaining pages.

The Crawler extracts the work's K-number from its title. This number is used to obtain the year and key of the work, which is located at:
http://www.classical.net/music/composer/works/mozart/

Since Mozarteum requires a valid session cookie, the crawler requires a valid session ID.

The data collected by the Crawler is stored in an SQLite3 database (not backwards compatible) [1].

In short, the steps are:

1. Grab a list of all categories from the site and store them in the database

[1] Do not try to open it with SQLite 1 or 2

2. Search for works in each category using Mozarteum's own search engine

3. For each work in the categories, find out which page range and what VSEP collection number (in the script called `group`) belongs to the work

4. Copy those pages from the Ripper storage to the Crawler storage into the folder ./work.id

C.0.3 Converting JPG Pages to BMP

The pages downloaded from Mozarteum is in JPEG format. Since JPEG is not supported by the OCR applications, the files are converted to monochrome BMP files.

This conversion is handled by `make_bmps.sh`, which makes sure not to reconvert JPEG files that already has a BMP version.

The conversion itself is done using Image Magick.

`make_bmps.sh` is a Shell script that works as follows:

```
for each folder in . ordered by int(name):
  for each file in folder ordered by int(name):
    if a file of name file.bmp exists:
      skip file
    convert file using Image Magick
```

Image Magick is used with the following parameters:

```
convert in.jpg -resize 2750x -sharpen 1 -compress none \
  -colors 2 out.bmp
```

The image is resized to the chosen width, sharpened and converting into a monochrome BMP file. The `-compress none` parameter ensures that no compression is used.

C.0.4 Converting BMP Pages to XML

This step is the most complicated as it depends on commercial applications that only run on Windows.

The conversion is started by the `start_bmp_to_xml.py` script which does the following:

The main thread starts 8 child threads and assigns each of them a set of Mozart works to process. Each child thread is also associated with an OCR-server (a virtual machine running Windows XP).

When a child thread needs to convert a BMP file to MusicXML, it proceeds as follows:

1. Move the file to your OCR-servers "input" folder
2. Wait until the file appears in your OCR-server's "done" folder
3. Read the MusicXML from the "done" folder

The conversion itself is not controlled by Python. If the file never shows up on the other side, the script blocks and manual forces are needed (however the remaining converter threads will go on).

The OCR-server works as follows:

1. Wait for BMP file in "input" folder
2. Move BMP file to "in use" folder
3. Call Liszt.exe (OCR engine) on BMP file to produce MRO file
4. Call SharpEye (OCR GUI Application) VBS script to convert MRO file to MusicXML (no, Liszt.exe cannot handle this)
5. Write MusicXML to "done" folder
6. Move BMP file from "in use" folder to "done" folder to show that work is done.

This process is not thread-safe when it comes to running multiple OCR-servers on the same virtual machine. The reason this works is that the client waits for the BMP file before reading the XML file and because the server closes the XML file before creating the BMP file.

Sometimes the OCR server fails to process the job. The most usual cause for this is that an error has occurred inside the SharpEye application. Such errors are ignored and the job must be reprocessed.

The OCR-server uses Visual Basic Scripting as described in section 2.4.1. It also has a Python part that uses the Liszt class in the util.ocr module. This module can convert BMP files to MRO using the SharpEye (Windows Only) binary Liszt.exe.

C.0.5 Converting XML Pages to Chords

Converting XML pages to chords is done by the `extract_chords.py` script.

The script selects all works that have no chords associated with them from the database. It then filters out the works where all pages have been converted to XML and extracts their chords. The result is stored in the database.

The script uses the extract_chords function from the music.chords module.

This function extracts the chords from a MusicXML file, by first parsing the file using the XMLScoreParser in conjunction with the DepthFirstParser from the parsers._xml module.

C.0.6 Calculating Entropy from Chords

This calculation is done by the script `calculate_entropies.py`

It works as follows:

1. Select all works that do not have entropies associated with them (from the database)
2. For each work, extract the chords
3. Use the statistics module to compute the entropies with runlength 1,2,3,4 and 5
4. Store the results in the database

All entropy computations are performed by the statistical.entropy module. This module contains two classes:

- Distribution
- Entropy

The Distribution class will create a distribution over N items and compute the probability for each item.

It implements the Python Dictionary, which means the probability of event e in sequence S can be computed by the Pythonic code:

`dist = Distribution(`S`)`
`prob = dist[`e`]`

By default, N will equal the length of S, but using a different N is needed when working with conditional probability.

The Entropy class has been modelled closely around the mathematical theory in order to make it easy to understand. Because of this, it is a bit slow. In order to speed it up a bit, the `P` function that computes unconditional entropies memorise its results for later use. These results are stored in the dictionary `P_cache`.

The Python code for computing conditional probability is:

```
# probability of seeing history
prob = self.P(history, map(lambda x: x[:-1],
# entropy of items preceding history
n_seq = []
for item in seq:
    if item[:-1] == history:
        n_seq.append(item[-1])
ent = sum (self.H(n_seq, item) for item in set(n_seq))
# conditional entropy
return prob * ent
```

These are the involved steps:

1. Compute the probability of the history
2. Compute the entropy of the elements that proceed the history
3. Multiply the results of 1) and 2).

C.0.7 Making Graphs from Entropy

Graphs are made by `make_graph.py`. The procedure is:

1. Select all works that has entropy values associated with them (from the database).
2. Generate graphs that display entropy values, standard deviation and average.
3. Store graphs in /var/www/images (for use with local website)

Graphs are constructed by the graphics.plotting module which uses the external module matplotlib.

matplotlib is a graph module for Python that uses MatLab syntax. This matplotlib module simplifies the plotting module, as it just adjusts the input to fit matplotlib's API and then calls this module to do the actual work.

Documentation of matplotlib can be found at:
http://matplotlib.sourceforge.net/

C.0.8 The Local Website

The local website is stored in /var/www. It consists of one html file (index.html) that presents the graphs located in /var/www/images.

Beware that index.html is automatically generated from index.html.tpl ; the template.

The template is processed by `update_site.py` that simply replaces a couple of variables.

The local website is *very* simple. It consists of two files, one HTML file and one template file (.tpl).

The template file is processed by `update_site_values.py`

Appendix D

Source Code

D.0.9 /src/make_graphs2.py

```
import sys, math, string

from config import *
from database import *

from cvxopt import solvers
from cvxopt.misc import matrix

from graphics.plotting import Plot

class Group:
    work_query = '''
␣␣␣␣SELECT␣Work.*,␣MIN(5000,Work.Chords)␣as␣chords_max5k,␣Entropy.value␣as␣entropy
␣␣␣␣FROM␣␣␣Work
␣␣␣␣␣␣␣␣␣␣JOIN␣Entropy
␣␣␣␣␣␣␣␣␣␣ON␣␣␣Entropy.work_id␣=␣Work.id
␣␣␣␣WHERE␣runlength␣=␣%s
␣␣␣␣ORDER␣BY␣%s
␣␣␣␣'''

    def __init__(self, group_by, field='entropy', order_by=['year', 'k_nr', 'id']):
        self.group_by = group_by
        self.field = field
        self.order_by = order_by

    def fun_get_field(self):
        return lambda x: getattr(x, self.field)
```

```
    def fun_get_group(self):
        if self.group_by == 'year':
            return lambda x: str(getattr(x, self.group_by))[-2:]
        return lambda x: getattr(x, self.group_by)

    def get_works(self, runlength):
        sql = Group.work_query % (runlength, ','.join([self.group_by] + self.order_by))
        return Work().query(sql)



GRAPH_PATH = '/var/www/images/'
ORDERS = (Group('year'),
                Group('category_id'),
                Group('chords'),
                Group('category_id', field='chords'),
                Group('category_id', field='chords_max5k'),
                )


def echo(s=None):
    sys.stdout.write(s or '\n')
    sys.stdout.flush()

def to_str(s):
    return ''.join(c for c in s if c in string.printable)


# calculate linear line through data points
def calc_line(data):
    if len(data) == 1:
        return (0, data[0][1] / float(data[0][0]))
    if len(data) == 2:
        p0, p1 = data
        a = (p1[1]-p0[1])/float(p1[0]-p0[0]) # (y1-y0)/(x0-x1)
        b = p0[1] - a*p0[0] # y0 - a*x0
        return (a, b)

    # generate optimization problem
    P = matrix(0., (2, 2) )
    q = matrix(0., (2, 1) )
    for x, y in data:
```

```
            P[0,0] += x**2 # + x^2 * a^2
            P[1,1] += 1 # + 1 * b^2
            P[0,1] += x # + x * ab
            P[1,0] += x # + x * ba
            q[0] -= 2*x*y # - 2xy * a
            q[1] -= 2*y # - 2y * b
        # solve problem
        sol = solvers.qp(2*P, q)
        return sol['x']

'''
points = [(1,6), (2, 4), (5, 2), (8, 2), (11, 4), (12, 6)]
a, b = calc_line(points)
print a,b
print sum([ (a*point[0] + b - point[1])**2 for point in points ])
print sum([ (4 - point[1])**2 for point in points ])
sys.exit()
'''

# calculate average
def avg_group(group, get_field):
    return float(sum(map(lambda w: get_field(w), group))) / len(group)


# calculate standard deviation
def dev_group(group, get_field):
    avg = avg_group(group, get_field)
    devs = map(lambda w: (get_field(w) - avg)**2, group)
    return math.sqrt(sum(devs) / len(group))


# plain and simple; just return values together with approx lines
def _plain(groups, get_field):
    shorts = 0
    xs = []
    values = []
    lines = []

    for id,group in groups:
        groupp = False
        if len(group) < 10:
            shorts += 1
        x_from = len(xs)
```

```
            xs += [id]*len(group)
            x_to = len(xs)
            g_values = map(lambda w: get_field(w), group)
            if x_to - x_from > 2:
                a, b = calc_line(zip(range(x_from, x_to), g_values))
                lines.append((x_from, x_to, a, b))
                # find outsiders
                sdev = math.sqrt(sum([ (a*(i+x_from)+b - get_field(work))**2
                                for i,work in enumerate(group) ])/len(group))
                for i, work in enumerate(group):
                    mydev = math.sqrt((a*(i+x_from)+b - get_field(work))**2)
                    if mydev > 2*sdev:
                        if not groupp:
                            print '>>group(%s)' % id
                            groupp = True
                        print '%4i␣|␣%.4f␣|␣%s' % (
                            work.k_nr, mydev, to_str(work.name)[:64])
            values += g_values
        a, b = calc_line(zip(range(0, len(xs)), values))
        if shorts > len(groups)/2:
            lines.append((0, len(xs), a, b))
        return xs, values, lines

# average
def _avg(groups, get_field):
    xs = [ id for id,_ in groups ]
    values = [ avg_group(group, get_field) for _,group in groups ]
    return xs, values, []


# standard deviation
def _dev(groups, get_field):
    xs = [ id for id,_ in groups ]
    devs = [ dev_group(group, get_field) for _, group in groups ]
    return xs, devs, []


# draw a graph based on xs, values (and errors)
def value_graph(xs, values, output, lines=[], errs=None):
    plot = Plot()
    if not errs:
        plot.flip_bar(xs, values)
    else:
```

```
            plot.error_bar(values, errs)
        for x_from, x_to, a,b in lines:
            plot.linear(x_from, x_to, a, b)
        plot.labels(xs)
        plot.set_fontsize(6)
        plot.save(output)
        plot.close()


# plot values according to fun and groups
def fun_graph(fun, get_field, groups, output):
    xs, values, lines = fun(groups, get_field)
    value_graph(xs, values, output, lines)


# draw a graph showing avg and standard deviation
def err_graph(groups, get_field, output):
    xs, avgs, _ = _avg(groups, get_field)
    _ , devs, _ = _dev(groups, get_field)
    value_graph(xs=xs, values=avgs, errs=devs, output=output)



# manual group by (we need to run functions on result)
def group_by(get_group, works):
    groups = []
    prev_g = None
    prev_l = []
    for work in works:
        group = get_group(work)
        if prev_g == None or group != prev_g:
            prev_g = group
            prev_l = []
            groups.append((group, prev_l))
        prev_l.append(work)
    return groups



# select works and entropy
for order in ORDERS:
    print 'order(%s.%s)' % (order.field, order.group_by)
    for runlength in (1, 2, 3, 4, 5):
```

```
        print '>runlength(%i)' % runlength
        works = order.get_works(runlength)
        groups = group_by(order.fun_get_group(), works)

        name = '%s.%s' % (order.field, order.group_by)
        for fun in (_plain, _avg, _dev):
            path = '%s/%s_%s_%i.png' % (GRAPH_PATH, name, fun.__name__, runlength)
            fun_graph(fun, order.fun_get_field(), groups, path)
        name += '__err'
        path = '%s/%s_%i.png' % (GRAPH_PATH, name, runlength)
        err_graph(groups, order.fun_get_field(), path)
        echo()
```

D.0.10 /src/start_bmp_to_xml.py

```
from config import *

import os, time
from threading import Thread


workdirs = ['0', '1', '2', '3']

WORK_PATH = '/home/shared/data/mozart/works/'

class Worker(Thread):
    def __init__(self, key, works):
        Thread.__init__(self)
        self.key = key
        self.works = works
        self.input = SHARPEYE_SERVER_INPUT % key
        self.inuse = SHARPEYE_SERVER_INUSE % key
        self.done = SHARPEYE_SERVER_DONE % key


    def inqueue_job(self, work, page):
        job = '%s__%s' % (work, page)
        from_path = '%s/%s/%s.bmp' % (WORK_PATH, work, page)
        to_path = '%s/%s.bmp' % (self.input, job)
        file(to_path, 'wb').write(file(from_path, 'rb').read())
        return job


    def get_result(self, job):
```

```
        jobpath = '%s/%s.bmp' % (self.done, job)
        jobxml = '%s.xml' % jobpath
        while not os.path.isfile(jobpath):
            time.sleep(1)
        time.sleep(1)
        data = file(jobxml).read()
        os.unlink(jobpath)
        os.unlink(jobxml)
        return data

    def job_ready(self, work, page):
        pagepath = '%s/%s/%s' % (WORK_PATH, work, page)
        return os.path.isfile('%s.bmp' % pagepath)

    def process_job(self, work, page):
        print '%s:␣%4s␣%4s' % (self.key, work, page)

        while not self.job_ready(work, page):
            time.sleep(1)

        job = self.inqueue_job(work, page)
        data = self.get_result(job)
        file('%s/%s/%s.xml' % (WORK_PATH, work, page), 'w').write(data)


    def process_work(self, work):
        pages = filter(lambda x: x.endswith('.jpg'), os.listdir('%s/%s' % (WORK_PATH, work)))
        for page in pages:
            if not os.path.isfile('%s/%s.xml' % (WORK_PATH, page)):
                self.process_job(work, page)

    def run(self):
        for work in self.works:
            self.process_work(work)


# start 8 threads, 2 on each sharpeye server
works = filter(lambda x: x != 'empty' and not '.' in x, os.listdir(WORK_PATH))
works.sort(key=lambda x: int(x))

partitions = [ list() for i in xrange(8) ]

i = 0
```

```
for work in works:
    # only add works where not all pages have been processed
    add_work = False
    for page in filter(lambda x: x.endswith('.jpg'), os.listdir('%s/%s' % (WORK_PATH, work))):
        if not os.path.isfile('%s/%s/%s.xml' % (WORK_PATH, work, page)):
            add_work = True
            break
    if add_work:
        partitions[i].append(work)
        i = (i + 1) % 8
    else:
        print 'skipping: %6s' % work

workers = [ Worker(key=str(i%4), works=partition) for i,partition in enumerate(partitions) ]


for worker in workers:
    worker.start()


for worker in workers:
    worker.join()
```

D.0.11 /src/graphics/__init__.py

D.0.12 /src/graphics/plotting.py

```
import matplotlib
matplotlib.use("Agg")


from pylab import *

class Plot:

    def __init__(self):
        figure(figsize=(12,8))
        self.width = 1.0

    def set_width(self, width):
        self.width = width

    def bar(self, heights, color='blue'):
```

```
        xs = tuple(i * self.width for i in xrange(len(heights)))
        return [ bar(x,h,width=self.width,color=color) for x,h in zip(xs, heights) ]

    def color_bar(self, heights, colors):
        xs = tuple(i * self.width for i in xrange(len(heights)))
        for (x, h), c in zip(zip(xs, heights), colors):
            bar(x, h, width=self.width, color=c)

    def error_bar(self, heights, errors):
        xs = tuple(i * self.width for i in xrange(len(heights)))

        return [ errorbar(x,h,error, fmt='x')
                 for (x,h),error in zip(zip(xs, heights), errors) ]

    def flip_bar(self, xs, values, _colors=['blue','red']):
        _colors = list(_colors)
        labels = self.labels(xs)
        prev = None
        colors = []
        color = 0
        for x, value in zip(xs, values):
            if prev == None or x != prev:
                color = (color + 1) % len(_colors)
                prev = x
            colors.append(_colors[color])
        self.color_bar(values, colors)

    def labels(self, in_labels, valign='center'):
        labels, prev = [], None
        for label in in_labels:
            if prev != None and label == prev:
                labels.append('')
                continue
            labels.append(label)
            prev = label
        xs = tuple(i*self.width for i in xrange(len(labels)))
        xticks( xs, labels, verticalalignment=valign )
        return labels

    def linear(self, x_from, x_to, a, b):
        xs, ys = [], []
        for x in xrange(x_from, x_to):
            xs.append(x)
```

```
            ys.append(a*x + b)
        plot(xs, ys, 'g', linewidth=2)

    def set_fontsize(self, size):
        map (lambda x: x.set_fontsize(size), axes().get_xticklabels())
        map (lambda x: x.set_fontsize(size), axes().get_yticklabels())

    def save(self, path):
        savefig(path)

    def close(self):
        close()


    @staticmethod
    def bar_graph(name_value_dict, graph_title='', output_name='bargraph.png'):
        figure(figsize=(8, 8)) # image dimensions
        title(graph_title, size='xx-small')

        # add bars
        for i, key in zip(range(len(name_value_dict)), name_value_dict.keys()):
            bar(i + 0.25, name_value_dict[key], color='red')

        # axis setup
        xticks(arange(.65, len(name_value_dict)),
                [('%s: %d' % (name, value)) for name, value in
                 zip(name_value_dict.keys(), name_value_dict.values())],
                size='xx-small')
        max_value = max(name_value_dict.values())
        tick_range = arange(0, max_value, (max_value / 7))
        yticks(tick_range, size='xx-small')
        formatter = FixedFormatter([str(x) for x in tick_range])
        gca().yaxis.set_major_formatter(formatter)
        gca().yaxis.grid(which='major')

        savefig(output_name)
```

D.0.13 /src/graphics/imaging.py

```
from PIL import Image


def sharpen(path_in, path_out):
    count = 0
```

```
    img = Image.open(path_in)
    pix = img.load()
    width,height = img.size

    for h in xrange(height-2):
        for w in xrange(width):
            oc = pix[w,h]
            nc = oc & pix[w,h+1] & pix[w,h+2]
            if oc != nc:
                count += 1
            pix[w,h] = nc
    img.save(path_out)
```

D.0.14 /src/mozateum/__init__.py

D.0.15 /src/mozateum/crawler.py

D.0.16 /src/process_emtpy.py

```
from config import *

import os, time
from threading import Thread


WORK_PATH = '/home/shared/data/mozart/works/empty/'

class Worker(Thread):
    def __init__(self, key, works):
        Thread.__init__(self)
        self.key = key
        self.works = works
        self.input = SHARPEYE_SERVER_INPUT % key
        self.inuse = SHARPEYE_SERVER_INUSE % key
        self.done = SHARPEYE_SERVER_DONE % key


    def inqueue_job(self, work, page):
        job = '%s__%s' % (work, page)
        from_path = '%s/%s/%s.bmp' % (WORK_PATH, work, page)
        to_path = '%s/%s.bmp' % (self.input, job)
        file(to_path, 'wb').write(file(from_path, 'rb').read())
```

```
        return job


    def get_result(self, job):
        jobpath = '%s/%s.bmp' % (self.done, job)
        jobxml = '%s.xml' % jobpath
        while not os.path.isfile(jobpath):
            time.sleep(1)
        time.sleep(1)
        data = file(jobxml).read()
        os.unlink(jobpath)
        os.unlink(jobxml)
        return data

    def job_ready(self, work, page):
        pagepath = '%s/%s/%s' % (WORK_PATH, work, page)
        return os.path.isfile('%s.bmp' % pagepath)

    def process_job(self, work, page):
        print '%s:␣%4s␣%4s' % (self.key, work, page)

        while not self.job_ready(work, page):
            time.sleep(1)

        job = self.inqueue_job(work, page)
        data = self.get_result(job)
        file('%s/%s/%s.xml' % (WORK_PATH, work, page), 'w').write(data)


    def process_work(self, work):
        pages = filter(lambda x: x.endswith('.jpg'), os.listdir('%s/%s' % (WORK_PATH, work)))
        for page in pages:
            if not os.path.isfile('%s/%s.xml' % (WORK_PATH, page)):
                self.process_job(work, page)

    def run(self):
        for work in self.works:
            self.process_work(work)


# start 8 threads, 2 on each sharpeye server
works = filter(lambda x: not '.' in x, os.listdir(WORK_PATH))
works.sort(key=lambda x: int(x))
```

```
worker = Worker('0', works)
worker.run()
```

D.0.17 /src/ocr/server.py

```
"""
Server will convert all files put in Z:/worksdirs/KEY/input to done folder
"""

from config import *

import os, time, threading

from util.ocr import Liszt, OCRException
from vbs import sharpeye_mro_to_xml

class Server:
    SHARPEYE_LOCK = threading.Lock()

    def __init__(self, key):
        self.key = key
        self.input = SHARPEYE_SERVER_INPUT % key
        self.inuse = SHARPEYE_SERVER_INUSE % key
        self.done = SHARPEYE_SERVER_DONE % key
        self.queue = set()
        for job in os.listdir(self.inuse):
            self.move_job(job, self.inuse, self.input)


    def finish_job(self, job, mro):
        # ensure only one sharpeye process at a time
        with Server.SHARPEYE_LOCK:
            xml = sharpeye_mro_to_xml(mro)
        self.move_job(job, self.inuse, self.done)
        xml_path = self.get_path(job + '.xml', self.done)
        file(xml_path, 'w').write(xml or '')


    def get_path(self, job, status):
        return '%s%s' % (status, job)

    def move_job(self, job, old, new):
        old_path = self.get_path(job, old)
```

```
        data = file(old_path, 'rb').read()

        new_path = self.get_path(job, new)
        file(new_path, 'wb').write(data)

        os.unlink(old_path)


    def get_more_jobs(self):
        jobs = os.listdir(self.input)
        self.queue.update(jobs)

    def get_job(self):
        while not self.queue:
            self.get_more_jobs()
            time.sleep(1)
        return self.queue.pop()

    def process_job(self):
        job = self.get_job()
        self.move_job(job, self.input, self.inuse)

        image_path = self.get_path(job, self.inuse)

        try:
            mro = Liszt.scan_image(image_path)
        except OCRException, e:
            mro = ''

        # run sharpeye routines in new thread
        self.finish_job(job, mro)


    def start(self):
        while True:
            self.process_job()
            time.sleep(1)
```

D.0.18 /src/ocr/__init__.py

D.0.19 /src/mozart_xml_left.py

```
from parsers import pdf
```

```
from parsers import ocr
import util.io

import os, os.path, sys

ROOT = 'Z:/data/mozart/'
BLOCK = 5

def isint(s):
    try:
        int(s)
    except:
        return False
    return True

names = sorted(filter(isint, os.listdir(ROOT)), key=lambda x: int(x))
names = reversed(names[:int(len(names))/2])
for name in names:
    path = ROOT + name
    if os.path.isdir(path):
        print '>> %s' % name
        pages_left = sorted(filter(lambda x: x.endswith('.bmp'), os.listdir(path)), key=lambda x: int(x.s
        while pages_left:
            tmp_dir = util.io.tmpdir()

            pages, pages_left = pages_left[:BLOCK], pages_left[BLOCK:]
            xml_file = '%s/%s.xml' % (path, pages[0])
            if len(pages) != BLOCK:
                break

            if os.path.isfile(xml_file):
                print 'skipping:', pages[0], pages[-1]
                continue
            print 'converting:', pages[0], pages[-1]

            # copy pages to tmp dir
            for page in pages:
                page_path = path + '/' + page
                data = file(page_path,'rb').read()
                file(tmp_dir + '/' + page.rjust(14, '0'), 'wb').write(data)

            # convert to music xml
            xml = pdf.bmps_to_xml(tmp_dir, check_type='first') or ''
```

```
        if xml:
            print '>> SUCCESS :)'
        else:
            print '>> FAILED! :('
        file(xml_file, 'w').write(xml)
        #util.io.rmdir(tmp_dir)
```

D.0.20 /src/scan_image.py

```
import sys
from util import ocr

print ocr.Liszt.scan_image_to_xml(sys.argv[1])
```

D.0.21 /src/database/__init__.py

```
from config import *

import os.path
from database.models import *
from pysqlite2 import dbapi2 as sqlite3

# import models
from models.category import Category
from models.composer import Composer
from models.work import Work
from models.chord import Chord
from models.entropy import Entropy

from models.composer_resource import ComposerResource
from models.work_resource import WorkResource

from models.resource import Resource
from models.resource_group import ResourceGroup

TABLES = ( Category, Composer, Work, Chord, Entropy,
           ComposerResource, WorkResource,
           Resource, ResourceGroup )


CONNECTION = None

def db_connect():
    # if no db, create it
```

```
    if not os.path.exists(DB_PATH):
        print 'could␣not␣find␣db,␣creating␣new'
        file(DB_PATH,'w').write('')
        db_create()
    global CONNECTION
    CONNECTION = CONNECTION or sqlite3.connect(DB_PATH)
    CONNECTION.row_factory = sqlite3.Row
    return CONNECTION

def db_cursor():
    db_connect()
    global CONNECTION
    return CONNECTION.cursor()

def db_create():
    map(lambda T: T().create(), TABLES)

def db_execute(sql, params=None):
    if not params:
        return db_cursor().execute(sql)
    params = tuple(params)
    return db_cursor().execute(sql, params)

def db_execute_one(sql, params=None):
    return db_execute(sql, params).fetchone()

def db_commit():
    db_connect().commit()

def db_update(sql, params=None):
    db_execute(sql, params)
    db_commit()
```

D.0.22 /src/database/test.py

```
from database import *

# creating new composer
from datetime import date
c = Composer(name='Bach', period_start=date.today(), period_stop=date.today())
c.save()
print 'ID:', c.id

c.name = 'stefan'
```

```
c.save()

# dump table
for c in Composer().query('SELECT * FROM Composer'):
    print c.id, c.name
```

D.0.23 /src/database/models/chord.py

```
from database.models import Model

class Chord(Model):
    TABLE = 'Chord'
    COLUMNS = ('id', 'symbol', 'work_id', 'position' )
    TYPES = ('INTEGER', 'VARCHAR(4)', 'INTEGER', 'INTEGER' )
```

D.0.24 /src/database/models/__init__.py

```
import database as db

class Model:
    def __init__(self, _row=None, **kwargs):
        setattr(self, self.COLUMNS[0], None)
        if _row:
            for key in self.COLUMNS:
                try:
                    _row[key] # check that key exists
                except IndexError,e:
                    raise IndexError(key)
            for key in _row.keys():
                setattr(self, key, _row[key])
        else:
            for key,value in kwargs.items():
                setattr(self, key, value)

    def create(self):
        print 'creating: %s %s' % (self, self.TABLE)
        columns = ('%s INTEGER PRIMARY KEY AUTOINCREMENT,' % self.COLUMNS[0]) + ','.j
        db.db_update('CREATE TABLE %s (%s)' % (self.TABLE, columns))

    def save(self):
        sql = ''
        if self.id:
            _set = ', '.join('%s=?' % col for col in self.COLUMNS[1:])
            sql = 'UPDATE %s SET %s WHERE id=%s' % (self.TABLE, _set, self.id)
```

```
        else:
            _set = ', '.join('?' for col in self.COLUMNS[1:])
            sql = 'INSERT INTO %s (%s) VALUES (%s)' % (self.TABLE, ', '.join(self.COLUMNS[1
        db.db_update(sql, (getattr(self, attr) for attr in self.COLUMNS[1:]))
        if not self.id:
            self.id = db.db_execute_one('SELECT last_insert_rowid() as RowId')['RowId']
        return self

    def query(self, query, params=None):
        return map(lambda row: self.__class__(_row=row), list(db.db_execute(query, params)))

    def query_one(self, query, params=None):
        row = db.db_execute_one(query, params)
        return row and self.__class__(_row=row) or None

    def __hash__(self):
        return hash(self.id)

    def __cmp__(self, other):
        return cmp(self.id, other.id)
```

D.0.25 /src/database/models/composer_resource.py

```
from database.models import Model

class ComposerResource(Model):
    TABLE = 'Composer_Resource'
    COLUMNS = ('id', 'composer_id', 'resource_id')
    TYPES = ('INTEGER', 'INTEGER' , 'INTEGER' )
```

D.0.26 /src/database/models/work.py

```
from database.models import Model

class Work(Model):
    TABLE = 'Work'
    COLUMNS = ('id', 'composer_id', 'name', 'year', 'key', 'category_id', 'pages', 'chords', 'k_nr')
    TYPES = ('INTEGER', 'INTEGER', 'VARCHAR(255)', 'INTEGER', 'VARCHAR(16)', 'INTEGER', '
, 'INTEGER')
```

D.0.27 /src/database/models/resource_group.py

```
from database.models import Model
```

```
class ResourceGroup(Model):
    TABLE = 'Resource_Group'
    COLUMNS = ('id', 'name', 'type' )
    TYPES = ('INTEGER', 'VARCHAR(255)', 'VARCHAR(16)' )
```

D.0.28 /src/database/models/resource.py

```
from database.models import Model

class Resource(Model):
    TABLE = 'Resource'
    COLUMNS = ('id', 'resource_group_id', 'uri', 'nodetype' )
    TYPES = ('INTEGER', 'INTEGER', 'VARCHAR(511)', 'VARCHAR(511)' )
```

D.0.29 /src/database/models/category.py

```
from database.models import Model

class Category(Model):
    TABLE = 'Category'
    COLUMNS = ('id', 'name' )
    TYPES = ('INTEGER', 'VARCHAR(255)' )
```

D.0.30 /src/database/models/entropy.py

```
from database.models import Model

class Entropy(Model):
    TABLE = 'Entropy'
    COLUMNS = ('id', 'work_id', 'method', 'runlength', 'value' )
    TYPES = ('INTEGER', 'INTEGER', 'VARCHAR(16)', 'INTEGER', 'REAL' )
```

D.0.31 /src/database/models/work_resource.py

```
from database.models import Model

class WorkResource(Model):
    TABLE = 'Work_Resource'
    COLUMNS = ('id', 'work_id', 'resource_id')
    TYPES = ('INTEGER', 'INTEGER', 'INTEGER' )
```

D.0.32 /src/database/models/composer.py

```
from database.models import Model
```

```
class Composer(Model):
    TABLE = 'Composer'
    COLUMNS = ('id', 'name', 'year_start', 'year_stop')
    TYPES = ('INTEGER', 'VARCHAR(255)', 'INTEGER', 'INTEGER', )
```

D.0.33 /src/music/__init__.py

D.0.34 /src/music/music.py

```
from gmpy import mpq
from parsers._xml import handler, traversers

class XMLScoreParser(handler.Handler):

    """Parse MusicXML document"""

    def __init__(self):
        self.__parts__ = {}
        self.__measures__ = {}
        self.__measure_numbers__ = []

        self.__measure__ = None
        self.__sign__ = None

        self.handlers = {
            'root' : self.root,
            'score-partwise' : self.score_partwise,
            'part-list' : self.part_list,
            'part' : self.part,
            'measure' : self.measure,
            'attributes' : self.attributes,
            'note' : self.note,

            'barline' : self.barline,
            'forward' : self.forward,
            'backup' : self.backup,
            }

    """ Node Handlers """
    def root(self, root, parent):
        return root.get_nodes()
```

```
    def score_partwise(self, score, parent):
        return score.get_nodes()

    def part_list(self, parts, parent):
        for part in parts.get_nodes():
            self.__parts__[part.get_attribute('id')] = []

    def part(self, part, parent):
        return part.get_nodes()

    # open new measure and add it to list
    def measure(self, xml_measure, parent):
        number = xml_measure.get_attribute('number')

        # start new measure (copy old if possible)
        if number in self.__measures__:
            measure = self.__measures__[number]
            measure.reset()
            measure.set_divisions(self.__measure__.divisions)
            self.__measure__ = measure
        else:
            self.__measure__ = self.__measure__ and self.__measure__.copy() or Measure()
            self.__measure__.set_number(number)
            self.__measures__[number] = self.__measure__
            self.__measure_numbers__.append(number)

        # traverse children
        return xml_measure.get_nodes()


    # parse attributes
    def attributes(self, attributes, parent):
        # divisions pr quarter
        if attributes['divisions']:
            self.__measure__.set_divisions(int(attributes['divisions'].get_content()))
        # key
        if attributes['key']:
            self.__measure__.set_key(int(attributes['key']['fifths'].get_content()))
        # time
        if attributes['time']:
            self.__measure__.set_time((int(attributes['time']['beats'].get_content()),
                    int(attributes['time']['beat-type'].get_content())))
        # check for percussion and interrupt if present
```

```
        if attributes['clef']:
            self.__sign__ = attributes['clef']['sign'].get_content()


    # barline ends measure
    def barline(self, barline, parent):
        #print 'barline: ', self.__measure__.number
        self.__measure__.end()

    # forward duration
    def forward(self, forward, parent):
        self.__measure__.forward(int(forward['duration'].get_content()))

    # backup duration
    def backup(self, backup, parent):
        self.__measure__.backup(int(backup['duration'].get_content()))

    # add note to measure
    def note(self, note, parent):
        # skip percussion notes
        if self.__sign__ == 'percussion':
            raise traversers.InterruptException()
        # skip abnormal notes
        if not note['duration']:
            return
        # skip unpitched notes (eg. drums/percussion)
        if note['unpitched']:
            return

        # extract duration
        duration = int(note['duration'].get_content())

        # pause
        if note['rest']:
            self.__measure__.forward(duration)
            return


        """  CHECKS DONE: NORMAL NOTE """

        xml_pitch = note['pitch']

        step = xml_pitch['step'].get_content()
```

```
        alter = xml_pitch['alter'] and int(xml_pitch['alter'].get_content()) or 0

        # note is part of a chord of previous notes
        if note['chord']:
            self.__measure__.add_note_to_previous(NoteInterval(step, alter), duration)
        # note is independent of previous notes
        else:
            self.__measure__.add_note(NoteInterval(step, alter), duration)
            self.__measure__.forward(duration)

        return note.get_nodes()


    """ Implementation get_handler """
    def get_handler(self, node, parent):
        if node.get_name() in self.handlers:
            return self.handlers[node.get_name()]


class Measure:
    def __init__(self, number=None, time=None, divisions=None, key=None):
        self.timeline = Timeline()
        self.now = mpq(0)
        self.length = 0

        self.set_number(number)
        self.set_time(time)
        self.set_divisions(divisions)
        self.set_key(key)


    """ set methods  """
    def set_number(self, number):
        self.number = number

    def set_time(self, time):
        self.time = time and mpq(time[0], time[1]) or None

    def set_divisions(self, divisions):
        self.divisions = divisions
        self.time_pr_dur = divisions and (mpq(1,4) / mpq(divisions)) or None

    def set_key(self, key):
```

```
        self.key = key


    """ note methods """
    def add_note(self, note, duration):
        note.duration = self.time_pr_dur * duration
        if self.now + note.duration > self.length:
            self.length = self.now + note.duration
        self.timeline.add_event(self.now, note)

    def add_note_to_previous(self, note, duration):
        note.duration = self.time_pr_dur * duration
        self.timeline.add_event_to_previous(note)

    """ time methods """
    def forward(self, duration):
        self.now += self.time_pr_dur*duration
        if self.now > self.length:
            self.length = self.now

    def backup(self, duration):
        self.now -= self.time_pr_dur*duration

    def skip_to(self, duration):
        self.now = self.time_pr_dur*duration

    """ get methods """
    def get_length(self):
        return self.timeline.get_length()

    """ end measure """
    def end(self):
        self.length = self.time

    """ reset start """
    def reset(self):
        self.now = mpq(0)

    """ check time """
    def check(self):
        return self.length == self.time

    """ copy measure """
```

```
    def copy(self):
        return Measure(number=self.number, time=self.time and (self.time.numer(), self.time.denom()))



class NoteInterval:
    notes_to_index = {
        'A': 0,
        'B': 2,
        'C': 3,
        'D': 5,
        'E': 7,
        'F': 8,
        'G': 10
        }

    def __init__(self, step, alter):
        self.note = (NoteInterval.notes_to_index[step] + alter) % 12
        self.start = None
        self.duration = None


class Timeline:
    def __init__(self):
        self.time = []
        self.prev_events = []
        self.prev_time = 0

    def add_event(self, start, event):
        event.start = start
        i = 0
        for i in xrange(len(self.time)):
            time, events = self.time[i]
            if time > start:
                break
            # time matches, append event
            if time == start:
                events.append(event)
                self.prev_events = events
                return
        # time passed by, insert event before
        events = [event]
        self.time.insert(i, (start, events))
```

```
        self.prev_events = events

    def add_event_to_previous(self, event):
        event.start = self.prev_time
        self.prev_events.append(event)

    def get_length(self):
        start, events = self.time[-1]
        return start + events[0].duration

    def count_events(self):
        return sum(map(lambda i: len(i[1]), self.time))
```

D.0.35 /src/music/chords.py

```
from parsers._xml import traversers
from music import XMLScoreParser


""" Identify chord from notes when in correct order"""
roots = ['A','Bb','H','C','Db','D','Eb','E','F','Gb','G','Ab']
def identify_chord(notes):
    chord = roots[notes[0]]
    if notes[0] + 3 in notes:
        chord += 'm'
    if notes[0] + 10 in notes:
        chord += '7'
    return chord

def find_chord(notes):
    chord, notes = [], set(notes)

    # test each note as being part of the chord
    for note in notes:
        _chord = _find_chord(note, notes)
        if len(_chord) > len(chord):
            chord = _chord
    return chord


def _find_chord(root, notes):
    chord = [root]
    for note in notes:
        if note in chord:
```

```
            continue
        interval = note-root
        new_notes = notes - set([note])
        for allowed in ((3,4), (6,), (10,)):
            for octave in (-12, 0, 12):
                if interval+octave in allowed:
                    chord += (_find_chord(note, new_notes))
    return chord


def extract_chords(parser):
    all_chords = []
    for measure_number in parser.__measure_numbers__:
        measure = parser.__measures__[measure_number]

        notes = []
        for (time, events) in measure.timeline.time:
            # remove expired notes
            notes = filter(lambda x: x.start+x.duration > time, notes)
            # add new notes
            notes += events
            # get chord in original position
            chord_notes = find_chord(map(lambda x: x.note, notes))
            # skip if less than 3 notes
            if len(chord_notes) < 3:
                continue
            # identify chord
            chord = identify_chord(chord_notes)
            all_chords.append(chord)
    return all_chords

def extract_chords_from_xml(path):
    handler = XMLScoreParser()
    score_parser = traversers.DepthFirstTraverser(handler)
    score_parser.traverse_xml_file(path)
    return extract_chords(handler)
```

D.0.36 /src/extract_chords.py

```
from config import *

import os
from music.chords import extract_chords_from_xml
from database import *
```

```
def get_work_path(work):
    return '%s%s/' % (WORK_PATH, work)

def get_page_path(work, page):
    return '%s%s' % (get_work_path(work), page)


def get_chords(work):
    pages = filter(lambda f: f.endswith('.jpg'), os.listdir(get_work_path(work)))
    page_xmls = []
    for page in pages:
        pagepath = get_page_path(work, page)
        if not os.path.isfile(pagepath + '.xml'):
            return None
        page_xmls.append(pagepath + '.xml')

    chords = []
    for page_xml in page_xmls:
        new_chords = extract_chords_from_xml(page_xml)
        chords.extend(new_chords)

    return chords


def add_chords(db_work, chords):
    for i, chord in enumerate(chords):
        db_chord = Chord(symbol = chord, position = i, work_id=db_work.id)
        db_chord.save()

work_query = '''
SELECT␣*␣FROM␣Work
WHERE␣NOT␣EXISTS
␣(SELECT␣*␣FROM␣Chord␣WHERE␣Work.id␣=␣Chord.work_id)
'''


works = Work().query(work_query)
for work in works:
    chords = get_chords(work.id)
    if chords:
        add_chords(work, chords)
        work.chords = len(chords)
```

```
        work.save()
        print '%s -> %s' % (work.id, len(chords))
```

D.0.37 /src/mozart_xml_right.py

```
from parsers import pdf
from parsers import ocr
import util.io

import os, os.path, sys

ROOT = 'Z:/data/mozart/'
BLOCK = 5

def isint(s):
    try:
        int(s)
    except:
        return False
    return True

names = sorted(filter(isint, os.listdir(ROOT)), key=lambda x: int(x))
names = names[int(len(names))/2:]
for name in names:
    path = ROOT + name
    if os.path.isdir(path):
        print '>> %s' % name
        pages_left = sorted(filter(lambda x: x.endswith('.bmp'), os.listdir(path)), key=lambda x: int(x.s
        while pages_left:
            tmp_dir = util.io.tmpdir()

            pages, pages_left = pages_left[:BLOCK], pages_left[BLOCK:]
            xml_file = '%s/%s.xml' % (path, pages[0])
            if len(pages) != BLOCK:
                break

            if os.path.isfile(xml_file):
                print 'skipping:', pages[0], pages[-1]
                continue
            print 'converting:', pages[0], pages[-1]

            # copy pages to tmp dir
            for page in pages:
                page_path = path + '/' + page
```

```
            data = file(page_path,'rb').read()
            file(tmp_dir + '/' + page.rjust(14, '0'), 'wb').write(data)

        # convert to music xml
        xml = pdf.bmps_to_xml(tmp_dir, check_type='first') or ''
        if xml:
            print '>> SUCCESS :)'
        else:
            print '>> FAILED! :('
        file(xml_file, 'w').write(xml)
        #util.io.rmdir(tmp_dir)
```

D.0.38 /src/make_graphs.py

```
import sys, math

from config import *
from database import *

from graphics.plotting import Plot

GRAPH_PATH = '/var/www/images/'


def make_graph(groups, prefix):
    entropies = [ list() for i in range(5) ]

    flag_colors = { True : 'blue', False : 'red' }

    flags = {}
    flag = False
    for group in groups:
        flag = not flag
        for work in group:
            flags[work.id] = flag
            work_entropies = Entropy().query('SELECT * FROM Entropy WHERE work_id=?',
                                             (work.id, ))
            for work_entropy in work_entropies:
                setattr(work_entropy, 'work', work)
                entropies[work_entropy.runlength-1].append(work_entropy)

    make_graph_runlength(flag_colors, flags, entropies, prefix)
```

```
def make_graph_runlength(flag_colors, flags, entropies, prefix):
    for runlength in (1,2,3,4,5):
        sys.stdout.write('.')
        sys.stdout.flush()

        plot = Plot()
        plot.set_width(8)

        heights = []
        colors = []
        for entropy in entropies[runlength-1]:
            heights.append( entropy.value > 0 and entropy.value or 0 )
            colors.append (flag_colors[flags[entropy.work_id]])

        plot.color_bars(heights, colors)

        xs = []
        for i, entropy in enumerate(entropies[runlength-1]):
            xs.append(str(getattr(entropy.work, order)))
        if order == 'year':
            xs = map(lambda x: x[-2:], xs)
        plot.labels(tuple(xs))
        plot.set_fontsize(6)

        plot.save(prefix + '_%i.png' % (runlength))
        plot.close()


def make_avg_graph(order, groups, prefix):
    averages = [ dict() for i in xrange(5) ]
    for group in groups:
        values = [ list() for i in xrange(5) ]
        group_name = ''
        for work in group:
            group_name = getattr(work, order)
            work_entropies = Entropy().query('SELECT␣*␣FROM␣Entropy␣WHERE␣work_id=?',
                                             (work.id,))
            for work_entropy in work_entropies:
                values[work_entropy.runlength-1].append(work_entropy.value)
        for i in xrange(5):
            averages[i][group_name] = len(values[i]) and sum(values[i]) / float(len(values[i])) or 0

    for runlength in xrange(5):
```

```
        plot = Plot()
        plot.bar(tuple(averages[runlength][year] for year in averages[runlength]))

        xs = tuple(name for name in averages[runlength])
        if order == 'year':
            xs = map(lambda xs: str(xs)[-2:], xs)
        plot.labels(xs)

        plot.set_fontsize(6)

        plot.save(prefix + '_%i.png' % (runlength+1))
        plot.close()

def make_spr_graph(order, groups, prefix):
    averages = [ dict() for i in xrange(5) ]
    for group in groups:
        values = [ list() for i in xrange(5) ]
        group_name = ''
        for work in group:
            group_name = getattr(work, order)
            work_entropies = Entropy().query('SELECT * FROM Entropy WHERE work_id=?',
                                             (work.id,))
            for work_entropy in work_entropies:
                values[work_entropy.runlength-1].append(work_entropy.value)
        for i in xrange(5):
            avg = len(values[i]) and sum(values[i]) / float(len(values[i])) or 0
            averages[i][group_name] = math.sqrt(sum(map(lambda x: (x - avg)**2, values[i])) / len(va

    for runlength in xrange(5):
        plot = Plot()
        plot.bar(tuple(averages[runlength][name] for name in averages[runlength]))

        xs = tuple(name for name in averages[runlength])
        if order == 'year':
            xs = map(lambda xs: str(xs)[-2:], xs)
        plot.labels(xs)

        plot.set_fontsize(6)

        plot.save(prefix + '_%i.png' % (runlength+1))
        plot.close()
```

```
def make_all_in_one(order, groups, prefix):
    averages = [ dict() for i in xrange(5) ]
    deviations = [ dict() for i in xrange(5) ]
    group_names = []
    for group in groups:
        values = [ list() for i in xrange(5) ]
        group_name = ''
        for work in group:
            group_name = getattr(work, order)
            work_entropies = Entropy().query('SELECT␣*␣FROM␣Entropy␣WHERE␣work_id=?',
                                             (work.id,))
            for work_entropy in work_entropies:
                values[work_entropy.runlength-1].append(work_entropy.value)
        for i in xrange(5):
            avg = len(values[i]) and sum(values[i]) / float(len(values[i])) or 0
            averages[i][group_name] = avg
            deviations[i][group_name] = math.sqrt(sum(map(lambda x: (x - avg)**2, values[i])) / len(
        group_names.append(group_name)

    for runlength in xrange(5):
        group_avg = averages[runlength]
        group_dev = deviations[runlength]

        plot = Plot()
        plot.error_bar([ group_avg[group] for group in group_names ],
                       [ group_dev[group] for group in group_names ])

        xs = tuple(name for name in averages[runlength])
        if order == 'year':
            xs = map(lambda xs: str(xs)[-2:], xs)
        plot.labels(xs)

        plot.set_fontsize(6)

        plot.save(prefix + '_%i.png' % (runlength+1))
        plot.close()



work_query = '''
SELECT␣*␣FROM␣Work
WHERE␣EXISTS␣(
```

```
␣␣SELECT␣*
␣␣FROM␣Entropy
␣␣WHERE␣Entropy.work_id␣=␣Work.id
)
ORDER␣BY␣%s,␣year,␣Work.id
'''

for order in ('year', 'category_id', 'chords'):
    sys.stdout.write(order + ':␣')
    groups = []

    if order == 'chords':
        prev = None
        i = -1
        for work in Work().query(work_query % order):
            if int(work.chords/50) != prev:
                groups.append([])
                prev = int(work.chords/50)
                i += 1
            groups[i].append(work)
    else:
        prev = None
        i = -1
        for work in Work().query(work_query % order):
            if getattr(work, order) != prev:
                groups.append([])
                prev = getattr(work, order)
                i += 1
            groups[i].append(work)

    make_graph(groups, GRAPH_PATH + order)
    make_avg_graph(order, groups, GRAPH_PATH + order + '_avg')
    make_spr_graph(order, groups, GRAPH_PATH + order + '_spr')
    make_all_in_one(order, groups, GRAPH_PATH + order + '_all')
    sys.stdout.write('\n')
```

D.0.39 /src/start_ocr_server.py

```
import sys
from ocr.server import Server

server = Server(key=sys.argv[1])
server.start()
```

D.0.40 /src/rip_mozart.py

```
import re, os, os.path
from url.urlgrabber.grabber import URLGrabber, URLGrabError

DOMAIN = 'http://dme.mozarteum.at'
URL_WORK = DOMAIN+'/DME/nma/scan.php?vsep=%i&l=2&p1=%i'
RE_PAGE = re.compile(r"src='([^']+)' alt='\-(\d+)\-'")

STORAGE = '/home/shared/data/mozart/'

URLGRABBER = URLGrabber(keepalive=0, retries=15)

def fetch_work(work_nr, pages_urls):
    if not os.path.isdir(STORAGE + str(work_nr)):
        os.mkdir(STORAGE + str(work_nr))
    for page in pages_urls:
        path = STORAGE + str(work_nr) + '/' + str(page) + '.jpg'
        if os.path.isfile(path):
            continue
        print 'grabbing:', work_nr, page
        try:
            data = URLGRABBER.urlread(DOMAIN+pages_urls[page])
        except URLGrabError,e:
            print e
            file('mozart_skipped', 'a').write('%6s %6s %s\n' % (str(work_nr), str(page), DOMAIN+p
            continue
        fd = file(path, 'w')
        fd.write(data)
        fd.close()

def fetch_work_pages(work_nr=0, start_page=0):
    print work_nr, start_page
    url = URL_WORK % (work_nr, start_page)
    data = URLGRABBER.urlread(url)
    page, pages = start_page, {}
    for (image_url, page) in RE_PAGE.findall(data):
        pages[int(page)] = image_url
    if len(pages) == 10:
        pages.update(fetch_work_pages(work_nr, int(page)+1))
    return pages

for work_nr in (127,135, 93, 79, 87):
```

```
    print '|| building page list for work nr %i..' % work_nr
    pages = fetch_work_pages(work_nr)
    print '|| fetching pages (%8i)..' % len(pages)
    fetch_work(work_nr, pages)
```

D.0.41 /src/update_page_count.py

```
from database import *
import os

WORK_PATH = '/home/shared/data/mozart/works/'

for work in Work().query('SELECT * FROM Work'):
    row = db_execute_one('SELECT COUNT(*) as count FROM Chord WHERE work_id=?', (wor
    pages = filter(lambda x: x.endswith('.jpg'), os.listdir('%s%s' % (WORK_PATH, work.id)))
    work.pages = len(pages)

    if work.chords != row['count']:
        work.chords = row['count']
        print work.id, work.pages, work.chords

    work.save()
```

D.0.42 /src/vbs/__init__.py

```
from config import *

import os, os.path
from util import io

def sharpeye(path):
    if path[-1] == '/':
        path = path[:-1]
    count = len(os.listdir(path))
    cmd = SHARPEYE_CMD % (path.replace('/', '\\'), count)
    os.system(cmd)
    if os.path.isfile('%s/AllPages.xml' % path):
        return file('%s/AllPages.xml' % path).read()
    return None

def sharpeye_mro_to_xml(mro_data):
    mro_path = io.tmpname(suffix='.mro')
    file(mro_path,'w').write(mro_data)
```

```
    mro_path = mro_path.replace('/', '\\')
    cmd = SHARPEYE_MRO_TO_XML_CMD % (mro_path)
    print cmd
    os.system(cmd)
    xml_path = '%s.xml' % mro_path
    if os.path.isfile(xml_path):
        xml_data = file(xml_path).read()
        os.unlink(xml_path)
        return xml_data
    return None


def close_window(title):
    os.system(CLOSE_WINDOW_CMD % title)
```

D.0.43 /src/statistics/__init__.py

D.0.44 /src/statistics/correlation.py

```
import math

def calc_correlation(pairs):
    N = len(pairs)
    if N < 2:
        return 0

    sum_x = sum_y = 0
    sum_xx = sum_yy = 0
    sum_xy = 0
    for x,y in pairs:
        sum_x += x
        sum_y += y
        sum_xx += x**2
        sum_yy += y**2
        sum_xy += x*y

    return (N*sum_xy - sum_x*sum_y) / \
        math.sqrt((N*sum_xx - sum_x**2) * (N*sum_yy - sum_y**2))
```

D.0.45 /src/statistics/entropy.py

```
import math
```

```
class Distribution(dict):

    def __init__(self, collection=None, N=None):
        dict.__init__(self)
        if collection:
            self.from_collection(collection, N=N)

    def equalize(self, count):
        self.count = count
        self.__getitem__ = lambda self, item : 1.0/self.count

    def from_collection(self, collection, N=None):
        self.count = N or len(collection)
        counts = {}
        for item in collection:
            counts[item] = counts.get(item, 0) + 1
        for item in counts:
            self[item] = counts[item] / float(self.count)

    def __getitem__(self, key):
        if not key in self and (key,) in self:
            return self[(key,)]
        if not key in self:
            print 'NOT FOUND! Here are the keys (count=%i):' % self.count
            print self.keys()
        return dict.__getitem__(self, key)

class Entropy:
    def __init__(self):
        self.dist = None
        self.dist_cache = {}

    # get runs from sequence
    def get_runs(self, items, N):
        res = list( tuple(items[i:i+N]) for i in xrange(len(items)-N+1) )
        return res

    # probability for item
    def P(self, item, seq=None):
        if not seq:
            return self.dist[item]
```

```
        key = tuple(seq)
        if key in self.dist_cache:
            dist = self.dist_cache[key]
        else:
            dist = Distribution(seq)
            self.dist_cache[key] = dist
        return dist[item]

    # probability conditioned by history
    def P_(self, seq, history=None):
        if not history:
            return self.P(seq, item)

        # probability of seeing history
        prob = self.P(history, map(lambda x: x[:-1], seq))

        # entropy of items preceding history
        n_seq = []
        for item in seq:
            if item[:-1] == history:
                n_seq.append(item[-1])
        ent = sum (self.H(n_seq, item) for item in set(n_seq))

        # conditional entropy
        return prob * ent


    # entropy value of item conditioned by hist
    def H(self, seq, item=None, history=[], base=2):
        N = len(history) + 1
        # unconditional (no history)
        if N == 1:
            prob = self.P(seq=seq, item=item)
            if prob > 0.0:
                return -1.0 * prob * math.log(prob, base)
        # conditional
        else:
            return self.P_(history=history, seq=seq)
        return 0.0

    # N is runlength, history is N-1
    def calc_entropy(self, items, N=1, base=2):
        entropy = 0.0
```

```
        if N > 1:
            runs = self.get_runs(items, N)
            for history in set(map(lambda x: x[:-1], runs)):
                entropy += self.H(runs, history=history, base=base)
        else:
            entropy = sum( self.H(items, item=item, base=base) for item in set(items))

        # entropy is NOT negative!
        assert entropy >= 0.0
        return entropy
```

D.0.46 /src/imslp/__init__.py

D.0.47 /src/imslp/crawler.py

```
import re, string, time
from threading import Thread
from urllib2 import urlopen
from database import *
from url.urlgrabber.grabber import URLGrabber

DOMAIN = 'http://imslp.org'

RE_COMPOSER = re.compile(r'<li><a href="([^"]+)" title="Category:[^"]+">([^<]+)</a></li>')
RE_COMPOSER_DTL = re.compile(r'<span class="mw-headline">([^<]+)</span></h2>\s*<p>\((

RE_ARTICLE = re.compile(r'<li><a href="([^"]+)" title="[^"]+">([^<]+)</a></li>')
RE_WORK = re.compile(r'>([^<]+)</a></b>[^"]*[^/]*([^"]+\.pdf)')

RE_CATEGORY = re.compile(r'intersect=([^"]+)"')

URL_COMPOSERS = '/wiki/Category:Composers&from=%s'

URL_GRABBER = None
def fetch(url):
    global URL_GRABBER
    URL_GRABBER = URL_GRABBER or URLGrabber(keepalive=1, retries=15)
    data = None
    while data == None:
        try:
            data = URL_GRABBER.urlread('%s%s' % (DOMAIN, url))
        except Exception,e:
            time.sleep(60)
```

```
    return data

class SimpleFetcher(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.pending = []
        self.completed = []

    def run(self):
        while True:
            if len(self.pending) == 0:
                continue
            key, url = self.pending.pop()
            data = fetch(url)
            self.completed.append(((key, url), data))

    def add(self, key, url):
        self.pending.append((key, url))

    def __getitem__(self, item):
        if (len(self.pending) == 0) and len(self.completed) == 0:
            raise IndexError()
        while len(self.completed) == 0:
            time.sleep(.1)
        return self.completed.pop()

class Crawler:

    def get_composers(self):

        # prepare fetcher
        fetcher = SimpleFetcher()

        # add urls to fetcher
        composers = {}
        for start in ('ABCDEFGHIJKLMNOPQRSTUVWXYZ'):
            data = fetch(URL_COMPOSERS % start)
            for (url, name) in RE_COMPOSER.findall(data):
                if not name in composers:
                    print 'adding:',name
                    fetcher.add(name, url)
                    composers[name] = None
        print 'starting fetcher..'
```

```
        # start fetcher
        fetcher.start()

        # process data
        for (key, url), data in fetcher:
            print key
            composers[key] = self.get_composer(url=url, data=data)
        return composers

    def get_composer(self, url, data=None):
        data = data or fetch(url)
        match = RE_COMPOSER_DTL.search(data)
        if not match:
            file('C:/music/composer_data','w').write(data)
            return
        name, _noise0, year_start, _noise1, year_stop = match.groups()
        c = Composer(name=name, url=url,
                     year_start=int(year_start), year_stop=int(year_stop))
        return c.save()

    def get_categories(self, composer):
        data = fetch(composer.url)
        file('C:/music/composer','w').write(data)
        for name in set(RE_CATEGORY.findall(data)):
            category = Category().query_one('SELECT␣*␣FROM␣Category␣WHERE␣name="%s"' %
            if not category:
                category = Category(name=name)
                category.save()
            yield category

    def get_works_by_category(self, composer, category):
        data = fetch(composer.url + '&intersect=%s' % category.name)
        return self.get_works_from_list(composer, category, data)

    def get_works(self, composer):
        for work in self.get_works_from_list(composer, None, fetch(composer.url)):
            yield work

    def get_works_from_list(self, composer, category, data):
        for url, name in RE_ARTICLE.findall(data):
            data = fetch(url)
            for work_name, url in RE_WORK.findall(data):
```

```
            work = db.db_execute_one('SELECT␣*␣FROM␣Work␣WHERE␣url=?', (url,))
            if work:
                yield work
                continue
            work = Work(name='%s␣−␣%s' % (name, work_name), url=url,
                        composer_id=composer.id, category_id=category and category.id or None
            work.save()
            yield work
```

D.0.48 /src/crawl_mozart.py

```
import re, os, os.path
from url.urlgrabber.grabber import URLGrabber, URLGrabError

DOMAIN = 'http://dme.mozarteum.at'
URL_WORK = DOMAIN+'/DME/nma/scan.php?vsep=%i&l=2&p1=%i'
RE_PAGE = re.compile(r"<img␣src='([^']+)'␣alt='\−(\d+)\−'")

STORAGE = '/home/shared/data/mozart/'

URLGRABBER = URLGrabber(keepalive=0, retries=15)

def fetch_work(work_nr, pages_urls):
    if not os.path.isdir(STORAGE + str(work_nr)):
        os.mkdir(STORAGE + str(work_nr))
    for page in pages_urls:
        path = STORAGE + str(work_nr) + '/' + str(page) + '.jpg'
        if os.path.isfile(path):
            continue
        print 'grabbing:', work_nr, page
        try:
            data = URLGRABBER.urlread(DOMAIN+pages_urls[page])
        except URLGrabError,e:
            print e
            file('mozart_skipped', 'a').write('%6s␣%6s␣%s\n' % (str(work_nr), str(page), DOMAIN+p
            continue
        fd = file(path, 'w')
        fd.write(data)
        fd.close()

def fetch_work_pages(work_nr=0, start_page=0):
    url = URL_WORK % (work_nr, start_page)
    data = URLGRABBER.urlread(url)
    page, pages = start_page, {}
```

```
    for (image_url, page) in RE_PAGE.findall(data):
        pages[int(page)] = image_url
    if len(pages) == 10:
        pages.update(fetch_work_pages(work_nr, int(page)+1))
    return pages

for work_nr in range(172,4096):
    print '|| building page list for work nr %i..' % work_nr
    pages = fetch_work_pages(work_nr)
    print '|| fetching pages..'
    fetch_work(work_nr, pages)
```

D.0.49 /src/crawl.py

```
from database import *
from imslp import crawler

crawler = crawler.Crawler()
#crawler.get_composers()

composers = list(Composer().query('''
        SELECT *
        FROM Composer
        WHERE id <
            (SELECT MIN(Composer.id)
            FROM Composer JOIN Work
                ON Work.composer_id = Composer.id
            )
    '''))

for i, composer in enumerate(reversed(composers)):
    try:
        print '%.8f: %5i, %s' % (float(i) / len(composers), composer.id, str(composer.name))
    except:
        pass
    categories = list(crawler.get_categories(composer))
    if categories:
        for j, category in enumerate(categories):
            list(crawler.get_works_by_category(composer, category))
    else:
        list(crawler.get_works(composer))
```

D.0.50 /src/mozart_bmp.py

```
#!/usr/bin/python

from parsers import pdf
import os, os.path
from parsers import ocr

for name in map(str, sorted(xrange(45,500))):
    path = '/home/shared/data/mozart/' + name
    if os.path.isdir(path):
        print '>> %s' % name
        for page in sorted(os.listdir(path)):
            print 'converting %s..' % page
            bmp_file = '%s/%s.bmp' % (path, page)
            xml = pdf.jpg_to_bmp('%s/%s' % (path, page), bmp_file)
```

D.0.51 /src/crawl_mozart_works.py

```
import os, re, urllib, urllib2
from database import *


STORAGE_PAGES = '/home/shared/data/mozart/'
STORAGE = '%sworks/' % STORAGE_PAGES


# fire up a browser and get yourself a valid session id
SESSION_ID = '1e74285fd5e8fc69e73e07a3f322a5d9'


# urls
URL_SEARCH = 'http://dme.mozarteum.at/DME/nma/nmapub_srch.php?l=2'
URL_CATEGORY = URL_SEARCH
URL_WORK_LIST = 'http://www.classical.net/music/composer/works/mozart/'

# regular expressions
RE_CATEGORY_DATA = re.compile(r'<select name="selGen" id="selGen" size="1">(.*?)</select>'

RE_CATEGORY = re.compile(r"<option\s*(selected)?\s*value='([^']+)'>")

RE_WORK = re.compile(r"\(\d+\)</td><td  style='padding-top:5px;' valign='middle' >([^<]+)</td

RE_K_NR = re.compile(r'(K|k)\.\s*(\d+)')

RE_WORK_INFO = re.compile(r'<td>K. (\d+)</td><td>[^<]*</td><td>(\d{4})[^<]*</td><t
```

```
def build_k_map():
    k_map = {}
    data = fetch(URL_WORK_LIST)
    for k_nr, year, key in RE_WORK_INFO.findall(data):
        k_map[k_nr] = (year, key)
    return k_map


def isint(s):
    try:
        int(s)
    except:
        return False
    return True


def fetch(url, post_data={}, session_id=SESSION_ID, retries=3):
    data = urllib.urlencode(post_data)
    request = urllib2.Request(url, data, {'cookie': 'PHPSESSID='+session_id})
    try:
        return urllib2.urlopen(request).read()
    except:
        if retries:
            return fetch(url, retries-1)
    return None


def get_categories():
    data = fetch(URL_CATEGORY)
    data = RE_CATEGORY_DATA.findall(data)[0]
    for _, category in RE_CATEGORY.findall(data):
        # skip numbers, as they are categories of categories
        if isint(category):
            continue
        yield category

def get_works(category):
    post_data = {'selLng': '2', 'selGen': category, 'ordby': '3', 'btnsubmit': 'Go'}
    data = fetch(URL_SEARCH, post_data)
    file('outputtmp','w').write(data)
    for title, group, page in RE_WORK.findall(data):
```

```
        yield title, group, page


def get_work_info(k_map, title):
    match = RE_K_NR.findall(title)
    k_nr = match and match[0][1] or None
    if k_nr != None and k_nr in k_map:
        return k_map[k_nr]
    if k_nr:
        print 'nothing for:', title
    return None


# get map of K-numbers -> (year, key=None)
k_map = build_k_map()

# create composer mozart if not exists
name = 'Wolfgang Amadeus Mozart'
composer = Composer().query_one('SELECT * FROM Composer WHERE name=?', (name,))
if not composer:
    composer = Composer(name=name, year_start=1756, year_stop=1791)
    composer.save()

# create resource_group "pages" if not present
page_folder = ResourceGroup().query_one(
    'SELECT * FROM Resource_Group WHERE name=?', ('page_folder',))
if not page_folder:
    page_folder = ResourceGroup(name='pagefolder', type='folder')
    page_folder.save()



# retrieve works for each category
print '>> crawl www'
categories = {}
groups = {}
count = 0
found = 0

for category in get_categories():
    print '>>', category
    categories[category] = []
```

```
    for title, group, page in get_works(category):
        if not group in groups:
            groups[group] = [int(page)]
        else:
            groups[group].append(int(page))
            groups[group].sort()
        count += 1
        info = get_work_info(k_map, title)
        if info:
            year, key = info
            categories[category].append( (title, (group, page), year, key) )
            found += 1

# insert categories into database
print '>> insert into database'
for category_name in categories:
    print '>>', category_name
    category = Category().query_one(
        'SELECT * FROM Category WHERE name=?', (category_name,))

    if not category:
        category = Category(name=category_name)
        category.save()

    works = categories[category_name]
    for idx, (title, (group, start_page), year, key) in enumerate(works):
        work = Work().query_one(
            'SELECT * FROM Work WHERE name=? AND year=? AND key=?', (title,year,key))

        work = Work(name=title, year=year, key=key,
                    composer_id=composer.id, category_id=category.id)
        work.save()


        # create resource
        path = '%s%i/' % (STORAGE, work.id)
        os.mkdir(path)

        # when does the next work start?
        pages = filter(lambda s: s.endswith('.jpg'), os.listdir(STORAGE_PAGES + group))
        pages = map(int, map(lambda x: x[:-1*len('.jpg')], pages))
        pages = filter(lambda i: i >= int(start_page), pages)
        pages.sort()
```

```
        # figure out when to stop
        stop_page = None
        for page in groups[group]:
            if page > int(start_page):
                stop_page = page-1
                break

        if stop_page:
            pages = filter(lambda i: i <= stop_page, pages)

        # check if all pages exists
        if stop_page:
            if not len(pages) == stop_page - int(start_page) + 1:
                print 'missing pages:', group, '%i-%i' % (int(start_page), stop_page)
                continue

        # lets copy the selected pages to the new resource location
        for page in list(pages):
            page_bin = file(STORAGE_PAGES + group + ('/%i.jpg' % page), 'rb').read()
            file(path + '%i.jpg' % page, 'wb').write(page_bin)

        # insert resources
        uri = 'file://data/mozart/works/%i/' % work.id
        resource = Resource().query_one(
            'SELECT * FROM Resource WHERE uri=?', (uri,))
        if resource:
            continue

        resource = Resource(resource_group_id=page_folder.id, uri=uri,
                            nodetype='folder')
        resource.save()
        WorkResource(work_id=work.id, resource_id=resource.id).save()


print 'got info for:', found
print 'missing for: ', count-found
```

D.0.52 /src/update_site_values.py

```
from database import *
from config import *
```

```
works = db_execute_one('''
SELECT␣COUNT(*)␣as␣count
FROM␣Work
WHERE␣EXISTS
␣␣(␣SELECT␣*
␣␣␣␣FROM␣␣␣Entropy
␣␣␣␣WHERE␣␣Entropy.work_id␣=␣Work.id
␣␣)
''')['count']

chords = db_execute_one('SELECT␣COUNT(*)␣as␣count␣FROM␣Chord')['count']

years = db_execute_one('''
SELECT␣MIN(Work.year)␣as␣min_year,␣MAX(Work.year)␣as␣max_year
FROM␣␣␣Work
WHERE␣␣EXISTS
␣(␣SELECT␣*␣FROM␣Chord␣WHERE␣Chord.work_id␣=␣Work.id␣)
''')

year_from = years['min_year']
year_to = years['max_year']
years = year_to - year_from

categories = db_execute_one('''
SELECT␣COUNT(*)␣as␣count
FROM␣␣␣category␣WHERE␣EXISTS
␣␣(SELECT␣*
␣␣␣FROM␣Work
␣␣␣WHERE␣Work.category_id␣=␣Category.id␣AND␣EXISTS
␣␣␣␣␣(SELECT␣*
␣␣␣␣␣␣␣FROM␣Chord
␣␣␣␣␣␣␣WHERE␣Chord.work_id␣=␣Work.id
␣␣␣␣␣)
␣␣);
''')['count']

vars = {
    '$$WORKS$$': works,
    '$$CHORDS$$': chords,
    '$$FROM$$': year_from,
    '$$TO$$': year_to,
    '$$YEARS$$': years,
    '$$CATEGORIES$$': categories
```

```
}

# generate new index file from template
tpl = file(WWW_ROOT + '/index.html.tpl').read()
for var in vars:
    tpl = tpl.replace(var, str(vars[var]))

file(WWW_ROOT + '/index.html', 'w').write(tpl)
```

D.0.53 /src/test.py

```
import sys
import os.path
from music import chords
from statistics.entropy import Distribution, Entropy

lst = ['A', 'Bb', 'B', 'C', 'Db', 'D', 'Eb', 'E', 'F', 'Gb', 'G', 'Ab']

def test(path='%s/%s' % (os.path.dirname(sys.argv[0]), r'samples/MozaVeilSample.xml')):
    all_chords = chords.extract_chords_from_xml(path)
    entropy = Entropy()
    print 'entropy:'
    import math
    for i in xrange(1,9):
        print 'runs %i: %f' % (i, entropy.calc_entropy(all_chords, N=i))

# run test
test(path='%s/%s' % (os.path.dirname(sys.argv[0]), r'samples/bicycle.xml'))
#len(sys.argv) > 1 and [ test(path) for path in sys.argv[1:]] or test()
```

D.0.54 /src/update_K_nr.py

```
import re
from database import *

RE_K_NR = re.compile(r'(K|k)\.\s*(\d+)')

for work in Work().query('SELECT * FROM Work'):
    match = RE_K_NR.findall(work.name)
    k_nr = match and match[0][1] or None
    work.k_nr = int(k_nr)
    work.save()
    print '%i -> %i' % (work.id, work.k_nr)
```

D.0.55 /src/parsers/_xml/__init__.py

```
from xml.parsers import expat

class Node:
    """
    XMLNode Python XML object
    """

    def __init__(self, name, attr={}, parent=None, content=None):
        self.__name__ = name
        self.__attrs__ = attr
        self.__content__ = content
        self.__parent__ = parent
        self.__nodes__ = []


    # properties
    def get_name(self):
        return self.__name__

    def get_content(self):
        return self.__content__

    def get_parent(self):
        return self.__parent__

    def set_content(self, content):
        self.__content__ = content


    # attributes
    def get_attribute(self, attr):
        return self.__attrs__.get(attr, None)

    def set_attribute(self, attr, value):
        self.__attrs__[attr] = value

    def get_attributes(self):
        return self.__attrs__


    # nodes
```

```
    def add_node(self, node):
        name = node.get_name()
        self.__nodes__.append(node)

    def get_nodes(self, name=None):
        if not name:
            for node in self.__nodes__:
                yield node
        for node in self.__nodes__:
            if node.get_name() == name:
                yield node

    def get_node(self, name):
        for node in self.get_nodes(name):
            return node

    # easy node access
    def __getitem__(self, name_or_index):
        if type(name_or_index) == int:
            return self.__nodes__[name_or_index]
        return self.get_node(name_or_index)

    # easy node check
    def __contains__(self, name):
        return bool(self.get_node(name))


    # string representation
    def __str__(self):
        return '<%s.XMLNode %s>' % (self.__module__, self.__name__)


class Parser:
    """
    Parse XML to XMLNode objects, preserving structure
    """
    def __init__(self):
        self.parser = expat.ParserCreate()
        self.parser.StartElementHandler = self.start_element
        self.parser.EndElementHandler = self.end_element
        self.parser.CharacterDataHandler = self.char_handler
```

```
    def parse_xml_file(self, path):
        return self.parse_xml(file(path).read())

    def parse_xml(self, xml):
        self.root = Node('root')
        self.current = self.root
        self.parser.Parse(xml)
        return self.root


    def start_element(self, name, attr):
        node = Node(name, attr, parent=self.current)
        self.current.add_node(node)
        self.current = node

    def end_element(self, name):
        self.current = self.current.get_parent()

    def char_handler(self, content):
        self.current.set_content(content)
```

D.0.56 /src/parsers/_xml/handler.py

```
class Handler():
    def set_root(self, root):
        self.root = root

    def get_handler(self, item, root, parent, node):
        raise NotImplementedError("You must implement __getitem__")
```

D.0.57 /src/parsers/_xml/traversers.py

```
from parsers._xml import Parser

class InterruptException(Exception):
    pass

class DepthFirstTraverser:
    """
    Depth-first XML Tree Traverser
    """
    def __init__(self, handler):
        self.handler = handler
```

```
        self.root = None

    def traverse_xml(self, xml_data):
        parser = Parser()
        parser.parse_xml(xml_data)
        self.set_root(parser.root)
        self.traverse(self.root)

    def traverse_xml_file(self, path):
        return self.traverse_xml(file(path).read())

    def set_root(self, node):
        self.root = node
        self.handler.set_root(self.root)

    def traverse(self, xml_node, parent=None):
        node_handler = self.handler.get_handler(node=xml_node, parent=parent)
        if not node_handler:
            return
        try:
            nodes = node_handler(xml_node, parent=parent)
            if not nodes:
                return
            for node in nodes:
                self.traverse(node, parent=xml_node)
        except InterruptException:
            pass
```

D.0.58 /src/parsers/__init__.py

```
class Handler():
    def set_root(self, root):
        self.root = root

    def get_handler(self, *args, **kwargs):
        raise NotImplementedError("You must implement get_handler(%s)" % kwargs.keys())
```

D.0.59 /src/parsers/ocr/__init__.py

D.0.60 /src/parsers/ocr/sharpeye.py

```
import vbs
```

```
def process_files(path):
    return vbs.sharpeye(path)
```

D.0.61 /src/parsers/mro/__init__.py

D.0.62 /src/parsers/mro/handler.py

```
from parsers import Handler
from traversers import DepthFirstTraverser

class MROHandler(Handler):

    def __init__(self):
        self.handlers = {
            'note': self.note,
            'chord': self.chord,
            'bar': self.bar,
            'stave': self.stave,
            }
        self.notes_by_pos = {}
        self.pos = None
        self.stave = None

    """ Implementation get_handler """
    def get_handler(self, node, parent):
        if node.get_name() in self.handlers:
            return self.handlers[node.get_name()]
        return self.default

    def default(self, node, parent):
        return node.get_nodes()

    def stave(self, node, parent):
        self.stave = (node.attributes['top'], node.attributes['left'])
        return node.get_nodes('bars')

    def bar(self, node, parent):
        return node.get_nodes('chords')

    def chord(self, node, parent):
        if 'flagposn' in node.attributes:
            self.pos = ','.join(reversed(node.attributes['flagposn'].split(',')))
```

```
            return node.get_nodes()

        def note(self, node, parent):
            pos = self.pos
            self.notes_by_pos[pos] = self.notes_by_pos.get(pos, []) + [node]
            return node.get_nodes()

P = {
       6: 'C',
       5: 'D',
       4: 'E',
       3: 'F',
       2: 'G',
       1: 'A',
       0: 'B',
     }
def getP(p):
    p = int(p)
    while p < 0 or p > 6:
        p = p < 0 and p + 7
        p = p > 6 and p - 7
    return P[p]

handler = MROHandler()
trav = DepthFirstTraverser(handler)
trav.traverse_mro_file('c:/output3.xml')
for key,value in sorted(handler.notes_by_pos.items(), key=lambda x: int(x[0].split(',')[0])):
    print key, '␣'.join( getP(note.attributes['p']) for note in value )
```

D.0.63 /src/parsers/mro/traversers.py

```
import re

class DepthFirstTraverser:
    """
␣␣␣␣Depth-first␣XML␣Tree␣Traverser
␣␣␣␣"""
    def __init__(self, handler):
        self.handler = handler
        self.root = None

    def traverse_mro(self, data):
        self.traverse(MRO(data=data))
```

```
    def traverse_mro_file(self, path):
        return self.traverse_mro(file(path).read())

    def set_root(self, node):
        self.root = node
        self.handler.set_root(self.root)

    def traverse(self, mro_node, parent=None):
        node_handler = self.handler.get_handler(node=mro_node, parent=parent)
        if not node_handler:
            return
        nodes = node_handler(mro_node, parent=parent)
        if not nodes:
            return
        for node in nodes:
            self.traverse(node, parent=mro_node)


class MRO:
    REGEX_HEAD = re.compile(r'SharpEyeMusicOCROutputFile')
    REGEX_TERM = re.compile(r'([^\s]+)\$?\s+(([^{"\s]+)|"([^"]|"")*")')
    REGEX_STRUCT = re.compile(r'([^\s]+)\s+{(.*})', re.DOTALL)
    REGEX_WS = re.compile(r'\s+')

    def __init__(self, name='root', data=None):
        self.blocks = (
                    (MRO.REGEX_HEAD, self.head),
                    (MRO.REGEX_TERM, self.term),
                    (MRO.REGEX_STRUCT, self.struct),
                    (MRO.REGEX_WS, self.ws),
                    )
        self.name = name
        self.attributes = {}
        self.nodes = []
        self.parse(data)

    def get_name(self):
        return self.name

    def get_nodes(self, name=None):
        return name and filter(lambda x: x.name==name, self.nodes) or self.nodes

    def head(self):
```

```
        return ''

    def ws(self):
        return ''

    def term(self, name, value, *args):
        self.attributes[name] = value
        return ''

    def struct(self, name, struct):
        data, count = '', 1
        vals = {'{': 1, '}': -1}
        for c in struct:
            count += vals.get(c, 0)
            if count == 0:
                break
            data += c
        self.nodes.append(MRO(name=name, data=data))
        res = struct[len(data)+1:]
        return res

    def parse(self, data):
        pos = 0
        while data:
            has_match = False
            for regex,handler in self.blocks:
                match = regex.match(data)
                if match:
                    has_match = True
                    pos += match.end()
                    data = handler(*match.groups()) + data[match.end():]
            if not has_match:
                raise Exception('No match!? (%i)\n%s' % (pos, data[:100]))

    def __repr__(self):
        retval = ''
        retval += '<%s %s>\n' % (self.get_name(), ''.join( '%s="%s" ' % (n,v) for (n,v) in self.attrib
        retval += '\n'.join(map(str, self.nodes))
        retval += '</%s>\n' % self.get_name()
        return retval
```

D.0.64 /src/parsers/pdf/__init__.py

```
import os, os.path
```

```
import util.ocr, util.io
import parsers.ocr.sharpeye

def pdf_page_to_bmp(pdf_file, bmp_file, page=0, args=[]):
    CMD = 'convert␣%s␣%s[%i]␣%s' % ('␣'.join(args), pdf_file, page, bmp_file)
    os.system(CMD)
    return os.path.isfile(bmp_file)

def jpg_to_bmp(jpg_file, bmp_file, args=['−resize␣2750x', '−sharpen␣1', '−compress␣none', '−colors␣
    CMD = 'convert␣%s␣%s␣%s' % (jpg_file, '␣'.join(args), bmp_file)
    os.system(CMD)
    return os.path.isfile(bmp_file)

def jpgs_to_bmps(jpg_dir, bmp_file):
    for i,fname in enumerate(os.listdir(jpg_dir)):
        path = bmp_file % str(i).rjust(8, '0')
        if jpg_to_bmp(jpg_dir + '/' + fname, path):
            yield path



def pdf_to_bmp(pdf_file, bmp_file=None, args=[]):
    bmp_file = bmp_file or pdf_file + '−%s.bmp'
    for page in xrange(1000):
        rjust_page = str(page).rjust(5, '0')
        path = bmp_file % rjust_page
        if not pdf_page_to_bmp(pdf_file, path, page, args):
            break
        yield path


def jpgs_to_xml(jpg_dir, xml_file=None):
    # get tmp work dir
    dir = util.io.tmpdir()
    bmp_file = '%s%s' % (dir, 'page−%s.bmp')
    # convert scores from jpg to bmps
    pages = list((jpgs_to_bmps(jpg_dir, bmp_file)))
    minimum = len(pages) * 0.8
    # find pages containing actual scores
    for i,page in enumerate(list(pages)):
        if not util.ocr.Liszt.is_score(page):
            pages.remove(page)
            os.unlink(page)
```

```
    # no scores, return
    if len(pages) < minimum or len(pages) < 1:
        return None
    # convert scores to music xml
    xml = parsers.ocr.sharpeye.process_files(dir)
    # delete tmp dir
    util.io.rmdir(dir)
    # return xml
    return xml


def score_to_bmp(pdf_file, bmp_file=None):
    return pdf_to_bmp(pdf_file, bmp_file, args=['-resize 2750x', '-sharpen 1', '-compress none', '-

def score_to_xml(pdf_file, xml_file=None):
    # get tmp work dir
    dir = util.io.tmpdir()
    bmp_file = '%s%s' % (dir, 'page-%s.bmp')
    # convert scores from pdf to bmps
    pages = list(score_to_bmp(pdf_file, bmp_file))
    xml = bmps_to_xml(dir)
    # delete tmp dir
    util.io.rmdir(dir)
    # return xml
    return xml

def bmps_to_xml(dir, check_type='all'):
    pages = tuple(dir + '/' + p for p in os.listdir(dir))

    if check_type == 'all':
        minimum = len(pages) * 0.8
        # find pages containing actual scores
        for i,page in enumerate(list(pages)):
            if not util.ocr.Liszt.is_score(page):
                pages.remove(page)
                os.unlink(page)
        # no scores, return
        if len(pages) < minimum:
            return None
    elif check_type == 'first':
        if not util.ocr.Liszt.is_score(pages[0]):
            return None
```

```
    # convert scores to music xml
    return parsers.ocr.sharpeye.process_files(dir)
```

D.0.65 /src/util/__init__.py

D.0.66 /src/util/io.py

```
import os, os.path
import random, string

__ALPHA__ = string.lowercase

def tmpname(path='C:/Temp/', prefix='', suffix='.tmp'):
    fname = None
    while (not fname or os.path.isfile('%s%s%s%s' % (path, prefix, fname, suffix))):
        fname = ''.join(random.sample(__ALPHA__, 8))
    return '%s%s%s%s' % (path, prefix, fname, suffix)

def tmpdir(path='C:/Temp/'):
    dirname = None
    while (not dirname or os.path.isdir(dirname)):
        dirname = '%s%s/' % (path, ''.join(random.sample(__ALPHA__, 8)))
    os.makedirs(dirname)
    return dirname

def rmdir(dir, recursive=True):
    if dir[-1] != '/':
        dir += '/'
    # delete all files and folders in dir
    if recursive:
        for name in os.listdir(dir):
            if os.path.isfile(dir+name):
                os.unlink(dir+name)
            elif os.path.isdir(dir+name):
                rmdir(dir+name)
    # dir should be empty now
    os.rmdir(dir)
```

D.0.67 /src/util/ocr.py

```
from config import *

import os, time
```

```
from io import tmpname
from threading import Thread
from vbs import close_window, sharpeye_mro_to_xml


class OCRException(Exception):
    pass

class Liszt:
    LISZT_CMD = '%(bin)s "%(input)s" "%(output)s" "%(config)s"'

    class Closer(Thread):
        def __init__(self, title):
            Thread.__init__(self)
            self.state = 'stopped'
            self.title = title

        def run(self):
            self.state = 'running'
            while self.state == 'running':
                close_window(self.title)
                time.sleep(.5)
            self.state = 'stopped'

        def stop(self):
            self.state = 'stopping'
            while self.state != 'stopped':
                time.sleep(.1)



    @staticmethod
    def scan_image(input_path):
        # get tmp file path
        output_file = tmpname(path=LISZT_TMP_PATH, suffix='.mro')

        # prepare command
        cmd = Liszt.LISZT_CMD % {'bin' : LISZT_BIN_PATH,
                                 'input' : input_path,
                                 'output': output_file,
                                 'config': LISZT_CFG_PATH}
        cmd = cmd.replace('/', '\\')
        print cmd
```

```
        # prepare to close stupid braindead popup warning from SharpEye
        closer = Liszt.Closer('Warning␣from␣SharpEye')
        closer.start()

        # run command
        retval = os.system(cmd)

        # stop closer
        closer.stop()

        # check output
        if not os.path.isfile(output_file):
            message = 'An␣error␣running␣OCR␣on␣%s.␣\n' + \
                      'Command␣was:␣\'%s\'\n' + \
                      'Return␣value:\n%s'
            raise OCRException(message % (input_path, cmd, retval))

        # read output xml
        data = file(output_file).read()
        # clean up
        os.unlink(output_file)
        return data

    @staticmethod
    def scan_image_to_xml(path):
        mro = None
        try:
            mro = Liszt.scan_image(path)
        except OCRException,e:
            return None
        return sharpeye_mro_to_xml(mro)



    @staticmethod
    def is_score(path):
        try:
            Liszt.scan_image(path)
        except OCRException,e:
            return False
        return True
```

D.0.68 /src/util/imaging.py

```
import os, time

class ImageMagick:
    IM_BIN_PATH = 'convert '

    @staticmethod
    def convert_pdf_to_bmp(path, page, output):
        if os.path.isfile(output):
            os.unlink(output)
        print '>' + ImageMagick.IM_BIN_PATH + '-density 300 "%s[%s]" "%s"' % (path,page,outp
        os.popen( ImageMagick.IM_BIN_PATH + '-density 300 "%s[%s]" "%s"' % (path,page,outpu
        return os.path.isfile(output)
```

D.0.69 /src/util/pdf.py

```
import imaging, io, os

class PDFReader:
    def __init__(self, path):
        self.path = path

    def get_pages_as_bmp(self):
        output = io.tmpname('C:/tmp/', '.bmp')
        i = 0
        while imaging.ImageMagick.convert_pdf_to_bmp(self.path, i, output):
            yield file(output,'rb').read()
            i += 1
        os.unlink(output)
```

D.0.70 /src/entropy_test.py

```
from database import *
from statistics import entropy


WORK_ID = 1004
RUNLENGTH = 5

chords = Chord().query('SELECT * FROM Chord WHERE work_id=?', (WORK_ID,))
symbols = map(lambda c: c.symbol, chords)

#print symbols
```

```
print '%4s: %5s chords' % (WORK_ID, len(symbols))

ent = entropy.Entropy()
value = ent.calc_entropy(symbols, N=RUNLENGTH)

print 'entropy:', value
```

D.0.71 /src/mozart.py

```
from parsers import pdf
import os, os.path
from parsers import ocr

for fname in os.listdir('mozart_pages'):
    if os.path.exists('mozart_pages/%s.xml' % fname):
        continue
    if os.path.isdir('mozart_pages/' + fname):
        print 'processing %s..' % fname
        try:
            xml = pdf.jpgs_to_xml('mozart_pages/%s' % fname) or ''
            file('mozart_pages/%s.xml' % fname, 'w').write(xml)
        except Exception,e:
            print 'Exception:', e
```

D.0.72 /src/supervisor.py

```
import commands, time, sys

# get vmware processes that use more than 50 percent CPU
CMD = 'top -bn1 | grep -i vmware | grep -E "S +([5-9][0-9]|100)"'


HOST = sys.argv[1]
MAX_IDLE = 30 * 60 # 30 minutes


def get_active_processes():
    data = commands.getoutput(CMD)
    pids = set()
    for line in data.split('\n'):
        if line:
            items = line.split(' ')
            if items[0]:
                pids.add(int(items[0]))
```

```
            else:
                pids.add(int(items[1]))
    return pids

def message(pid, status):
    subject = 'AoMC_status: [%s] VMware: %s -> %s' % (HOST, str(pid).rjust(5, '0'), status)
    commands.getoutput('zsh /home/shared/code/musicXML/src/sendmail.sh "%s"' % subject)

def supervise():
    processes = {}
    while True: # never surrender!
        for pid in get_active_processes():
            if pid in processes:
                status, _ = processes[pid]
                if status == 'inactive':
                    message(pid, 'active')
                print pid, 'active'
                processes[pid] = ('active', time.time())
            else:
                processes[pid] = ('active', time.time())
                message(pid, 'new')

        for pid in processes:
            status, last_seen = processes[pid]
            if status == 'active' and time.time() - last_seen > MAX_IDLE:
                message(pid, 'inactive')
                processes[pid] = ('inactive', last_seen)
        time.sleep(1)

supervise()
```

D.0.73 /src/url/__init__.py

D.0.74 /src/mozart_xml_back.py

```
from parsers import pdf
from parsers import ocr
import util.io

import os, os.path, sys

ROOT = 'Z:/data/mozart/'
BLOCK = 5
```

```
def isint(s):
    try:
        int(s)
    except:
        return False
    return True

names = sorted(filter(isint, os.listdir(ROOT)), key=lambda x: int(x))
for name in reversed(names):
    path = ROOT + name
    if os.path.isdir(path):
        print '>> %s' % name
        pages_left = sorted(filter(lambda x: x.endswith('.bmp'), os.listdir(path)), key=lambda x: int(x.s
        while pages_left:
            tmp_dir = util.io.tmpdir()

            pages, pages_left = pages_left[:BLOCK], pages_left[BLOCK:]
            xml_file = '%s/%s.xml' % (path, pages[0])
            if len(pages) != BLOCK:
                break

            if os.path.isfile(xml_file):
                print 'skipping:', pages[0], pages[-1]
                continue
            print 'converting:', pages[0], pages[-1]

            # copy pages to tmp dir
            for page in pages:
                page_path = path + '/' + page
                data = file(page_path,'rb').read()
                file(tmp_dir + '/' + page.rjust(14, '0'), 'wb').write(data)

            # convert to music xml
            xml = pdf.bmps_to_xml(tmp_dir, check_type='first') or ''
            if xml:
                print '>> SUCCESS :)'
            else:
                print '>> FAILED! :('
            file(xml_file, 'w').write(xml)
            #util.io.rmdir(tmp_dir)
```

D.0.75 /src/config.py

```
import os


""" PATH TO SHARED SAMBA DRIVE (windows != posix) """
SHARED = None
if os.name == 'posix':
    SHARED = '/home/shared/'
else:
    SHARED = 'Z:/'


""" DATABASE """
DB_PATH = SHARED + 'data/musicdb.sqlite3'

""" PATH TO STORAGE """
WORK_PATH = SHARED + 'data/mozart/works/'

""" PATH TO WWW """
WWW_ROOT = '/var/www/'


""" SHARPEYE """
SHARPEYE_CMD = \
    'CScript.exe "%scode/musicXML/src/vbs/sharpeye.vbs" "%%s" %%i' % SHARED

SHARPEYE_MRO_TO_XML_CMD = \
    'CScript.exe "%scode/musicXML/src/vbs/sharpeye_mro_xml.vbs" "%%s"' % SHARED

CLOSE_WINDOW_CMD = \
    'CScript.exe "%scode/musicXML/src/vbs/close_window.vbs" %%s' % SHARED


""" SHARPEYE SERVER """
SHARPEYE_SERVER_INPUT = '%sworkdirs/%%s/input/' % SHARED
SHARPEYE_SERVER_INUSE = '%sworkdirs/%%s/inuse/' % SHARED
SHARPEYE_SERVER_DONE = '%sworkdirs/%%s/done/' % SHARED


""" LISZT (Windows-only) """
LISZT_BIN_PATH = 'C:/SharpEye2/liszt.exe'
LISZT_CFG_PATH = 'C:/SharpEye2/configfile'
LISZT_TMP_PATH = 'C:/SharpEye2/tmp/'
```

D.0.76 /src/calc_entropy.py

```
import os
from statistics.entropy import Distribution, Entropy
from music.chords import extract_chords_from_xml
from graphics import plotting

chords = {}
all_chords = []

fnames = filter(lambda x: x.endswith('.xml'), os.listdir('samples/'))
for fname in fnames:
    print 'analysing %s..' % fname

    chords[fname] = extract_chords_from_xml('samples/%s' % fname)
    all_chords += chords[fname]

    print 'chords found: %i' % len(chords[fname])

print 'total number of chords: %s' % len(set(all_chords))

# calc entropies
entropies = []
for fname in chords:
    print fname
    ent = Entropy()
    entropies.append((ent.calc_entropy(chords[fname]),fname))

# display sorted
entropies.sort()
for item in entropies:
    print '%.4f  %s' % (item)

"""
# make bar plot
plot = plotting.Plot()
plot.set_width(8)
plot.bar([ value for value, _ in entropies])
plot.labels(map(lambda x: 'MZ'+x[4:], fnames))
plot.save('entropies.png')
"""
```

D.0.77 /src/pdf.py

```
from parsers import pdf
import os, os.path
from parsers import ocr

for name in os.listdir('C:/bach/'):
    if os.path.exists('C:/bach/%s.xml' % name):
        continue
    if name.lower().endswith('.pdf'):
        print 'processing␣%s..' % name
        try:
            xml = pdf.score_to_xml('C:/bach/%s' % name) or ''
            file('C:/bach/%s.xml' % name, 'w').write(xml)
        except Exception,e:
            print 'Exception:', e
```

D.0.78 /src/mozart_xml.py

```
from parsers import pdf
from parsers import ocr
import util.io

import os, os.path, sys

ROOT = 'Z:/data/mozart/'
BLOCK = 5

def isint(s):
    try:
        int(s)
    except:
        return False
    return True

names = sorted(filter(isint, os.listdir(ROOT)), key=lambda x: int(x))
for name in names:
    path = ROOT + name
    if os.path.isdir(path):
        print '>>␣%s' % name
        pages_left = sorted(filter(lambda x: x.endswith('.bmp'), os.listdir(path)), key=lambda x: int(x.s
        while pages_left:
            tmp_dir = util.io.tmpdir()

            pages, pages_left = pages_left[:BLOCK], pages_left[BLOCK:]
            if len(pages) != BLOCK:
```

```
            break

        xml_file = '%s/%s.xml' % (path, pages[0])

        if os.path.isfile(xml_file):
            print 'skipping:', pages[0], pages[-1]
            continue
        print 'converting:', pages[0], pages[-1]

        # copy pages to tmp dir
        for page in pages:
            page_path = path + '/' + page
            data = file(page_path,'rb').read()
            file(tmp_dir + '/' + page.rjust(14, '0'), 'wb').write(data)

        # convert to music xml
        xml = pdf.bmps_to_xml(tmp_dir, check_type='first') or ''
        if xml:
            print '>> SUCCESS :)'
        else:
            print '>> FAILED! :('
        file(xml_file, 'w').write(xml)
        #util.io.rmdir(tmp_dir)
```

D.0.79 /src/calculate_entropies.py

```
import os, sys
from database import *
from statistics import entropy


work_query = '''
SELECT * FROM Work
WHERE EXISTS
  ( SELECT *
    FROM    Chord
    WHERE   Chord.work_id = Work.id )
AND NOT EXISTS
  ( SELECT *
    FROM    Entropy
    WHERE   Entropy.work_id = Work.id )
ORDER BY chords
'''
```

```
works = Work().query(work_query)
for work in works:
    chords = Chord().query(
        'SELECT␣*␣FROM␣Chord␣WHERE␣Chord.work_id=?␣ORDER␣BY␣position', (work.id,))
    symbols = map(lambda c: c.symbol, chords)

    sys.stdout.write('%4s␣(%5i):␣' % (work.id, len(symbols)))

    ent = entropy.Entropy()
    for runlength in (1,2,3,4,5):
        sys.stdout.write('.')
        sys.stdout.flush()

        value = ent.calc_entropy(symbols, N=runlength)
        db_entropy = Entropy(work_id=work.id, method='simple',
                             runlength=runlength, value=value)
        db_entropy.save()
    sys.stdout.write('\n')
```

D.0.80 /src/stats.py

```
import os, pickle
from graphics import plotting
from music import chords
from statistics.entropy import Entropy

stats = {}

try:
    stats = pickle.loads(file('dumped').read())
except Exception,e:
    print e

print stats
if not stats:
    files = filter(lambda x: x.endswith('.pdf.xml'), os.listdir('C:/bach/'))
    files = filter(lambda x: not x.endswith('_001.pdf.xml'), files)
    for i,name in enumerate(files):
        path = 'C:/bach/%s' % name
        print 'processing␣%4s␣of␣%s:␣%s..' % (i, len(files), name)

        all_chords = chords.extract_chords_from_xml(path)
        if len(all_chords) == 0:
```

```
            continue

        file('graf-pdfs/%s' % name[:-4],'wb').write(file(path[:-4], 'rb').read())

        entropy = Entropy()
        stats[name] = []
        for i in xrange(1,4):
            stats[name].append(entropy.calc_entropy(all_chords, N=i))

file('dumped','w').write(pickle.dumps(stats))

plot = plotting.Plot()
colors = ('blue', 'red', 'green')

for i in xrange(3):
    plot.set_width(1)
    plot.bar([ stats[k][i] for k in stats ], color=colors[i])

plot.labels(range(len(stats)))
plot.save('barplot.png')

print stats
```