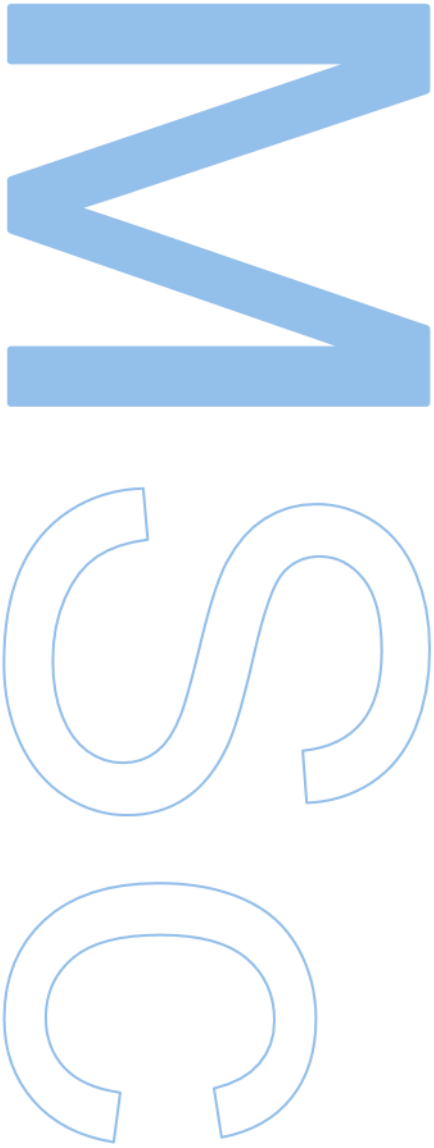
Characterizing Music through Complex Networks



Manuel Bravo Silva Lamas
Master's degree in Computer Science
Computer Science Department
2022

Supervisor
Pedro Manuel Pinto Ribeiro, Assistant Professor
Faculty of Science, University of Porto


U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Sworn Statement

I, Manuel Bravo Silva Lamas, enrolled in the Master Degree in Computer Science at the Faculty of Sciences of the University of Porto hereby declare, in accordance with the provisions of paragraph a) of Article 14 of the Code of Ethical Conduct of the University of Porto, that the content of this dissertation reflects perspectives, research work and my own interpretations at the time of its submission.

By submitting this dissertation, I also declare that it contains the results of my own research work and contributions that have not been previously submitted to this or any other institution.

I further declare that all references to other authors fully comply with the rules of attribution and are referenced in the text by citation and identified in the bibliographic references section. This dissertation does not include any content whose reproduction is protected by copyright laws.

I am aware that the practice of plagiarism and self-plagiarism constitute a form of academic offense.

Manuel Bravo Silva Lamas

September 28, 2022

Abstract

Complex networks can be seen in many domains, music being one of them. We seek to characterize music by studying complex networks' topology.

Music is not easily defined, as there are many aspects to it, with some more easily identified than others. The most common method to analyse music is through sound as a final result. In this work we look at it through a symbolic representation, via the standard MIDI format. Unlike in an audio representation, where the audio waves are recorded, in a symbolic representation we have essentially a generalization of the common musical score. By looking at music through its most basic elements, which are the notes, we attempt to capture information that is not available in other formats.

In order to achieve it, one mapping from songs to networks is presented. From the mapped networks, features are chosen with the intent of characterizing the songs. To evaluate how well the arrived feature set works, a set of synthetic MIDIs was designed, created, and then analysed. We follow with an analysis of a dataset of chosen (non-synthesised) songs. In both analysis, we describe the distinctions that can be made between certain groups of songs, or synthetic examples, when comparing specific features (or set of features) values.

Finally, we present as a proof-of-concept the clustering results of the mentioned music dataset, as a potential application of our proposed mapping and feature set.

Keywords: Complex Networks, Music, Characterization, MIDI

Resumo

Redes complexas podem ser vistas em vários domínios, música sendo um deles. Nós pretendemos caracterizar música através do estudo da topologia de redes complexas.

Música não é algo facilmente definido, pois contém vários aspetos, alguns mais facilmente identificáveis do que outros. O método mais comum para analisar música é através do som como produto final. Neste trabalho analisamos música através de uma representação simbólica, via o formato standard MIDI. Ao contrário de uma representação audio, onde as ondas de som são gravadas, numa representação simbólica nós temos essencialmente uma generalização de uma pauta musical comum. Observando a música através dos seus elementos mais simples, as notas, tentamos capturar informação que não está disponível noutros formatos.

Para o conseguir, é apresentado um mapeamento de músicas em redes. Dessas redes mapeadas são escolhidas características com o intuito de caracterizar as músicas. Para avaliar o desempenho do conjunto de caracteristicas escolhido, um conjunto de MIDIs sintéticos foi desenhado, criado e analisado. Seguimos com uma análise de um conjunto de músicas (não sintéticas). Em ambas as análises nós descrevemos as distinções que podem ser feitas entre certos grupos de músicas, ou exemplos sintéticos, quando comparando os valores de uma característica específica (ou conjunto de caracteristicas).

Finalmente, apresentamos como prova de conceito os resultados de uma partição efetuada através do mapeamento e conjunto de caracteristicas escolhidos, como potencial aplicação dos mesmos.

Palavras-chave: Redes Complexas, Música, Caracterização, MIDI

Acknowledgments

I want to thank my supervisor Pedro Ribeiro, that kept me interested on the next step, and was immensely valuable with his inputs whenever I reached a dead end. Offering multiple alternatives for how to follow up.

I am grateful for my family and friends that supported me during the whole period, waiting patiently for me to complete it.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Networks as a tool are quite powerful and flexible, they can be used to model and analyse systems of many areas. Some network science specific areas of study are: roles [11], communities [3], motifs [18], and higher-order networks [25].

However, Network Science has recently appeared as a multidisciplinary domain to study networks incorporating a variety of topics and domains (graph theory, statistical mechanics, data mining, data visualization, etc) [1]. Since then, it has been successfully applied to a multitude of domains [5] such as social [24], road [19], biology networks [6] or even computer science networks [22].

Music as also received some attention from a network science perspective. Most of the work tackles aspects such as social networks (e.g. of musicians [8] or composers [17, 20]) or more classical data mining tasks (e.g. music recommendation using lyrics [10]). In this work we are more concerned with actually modelling the music itself and the melodies and notes. There exists already some related work [7, 23], although not as much studied through the lenses of complex networks as other subjects. Some existing studies involve revealing how a genre evolved overtime [21], generating appealing music [15, 16], and studying music as a higher-order network [23]. Nevertheless, many aspects and approaches have yet to be tested.

1.1 Goals and Contributions

Our goal, with this thesis, is to ascertain whether a feature (or set of features), obtained through "common" network science metrics, can be used to capture simple musical aspects. Moreover, we seek to realize whether those are sufficient to differentiate between sets of songs with different characteristics or different genres.

The main contributions described in the thesis are:

- A methodology able to map songs into networks, describing exactly what MIDI attributes we capture with our specific mapping.

- The introduction of a feature set able to characterize songs through their mapped networks.
- An experimental study that illustrates and explains the potential of each feature as a way to individually, or as a subset of the feature set, capture specific musical aspects.
- A test of the feature set on a "real" music dataset with songs from 3 different genres (classic, pop, and rock)
- As a proof-of-concept, a method to partition the music dataset via k-means with our proposed feature set as attributes

All analysis was done with Python, particularly using the packages `NetworkX`[1] [12] for handling networks, and `Mido`[2] for handling MIDI files. The code used to make all the analysis present here, in addition to the datasets mentioned, are all available at GitHub.[3]

1.2 Thesis Structure

This thesis is divided in the following chapters. We now provide a short description of each.

Chapter 1 - Introduction Briefly explains the usefulness of network science as a domain, as well as our intentions and goals. Giving a short description of each chapter's main points.

Chapter 2 - Preliminaries Presents definitions for networks, its basic concepts and metrics, as well as some musical concepts, which will be used on following chapters.

Chapter 3 - Songs to Networks Discusses how we obtain the information from the music files, in the MIDI format. As well as how we process it, and how we map each song to a network that will be used to characterize it. It also describes what attributes existing in the MIDI format it does capture, as well as which ones it does not.

Chapter 4 - Analysis Describes exactly which features will be part of our feature set to analyse songs. Presenting a dataset of synthesized MIDI examples, designed and created to test our feature set. It then proceeds to showing the results of the feature set for the dataset, describing it and highlighting differences between the various examples identified through the chosen features. As well as testing the feature set on a dataset of chosen songs, and presenting a partition of that same dataset in clusters as a proof-of-concept to glimpse at the capabilities of the proposed feature set.

Chapter 5 - Conclusions Concludes the thesis by presenting our main contributions, as well as our ideas and possible approaches to be undertaken in future work.

[1] `https://networkx.org/`
[2] `https://mido.readthedocs.io/en/latest/`
[3] `https://github.com/manuellamas/Music_and_Networks`

Chapter 2

Preliminaries

In this chapter we will present basic complex networks, musical concepts and metrics, as well as the associated terminology, which will be used throughout the rest of this thesis. We will begin by explaining what a network (or graph) is, and then proceed to explaining what features can characterize a network's topology, illustrating each of them with the aid of a few toy networks. Similarly, we will explain basic musical concepts relevant to our work. Starting by what is a musical score, and how it represents music. To then defining what is a note, and how it represents duration, and sound (pitch).

2.1 Complex Networks

Complex networks can be found in a variety of domains, as is proof of its many applications. Music being something that is present in most people's daily lives, it is only natural that it is studied as well. There have been many approaches to studying music, a few already resorting to Complex Networks.

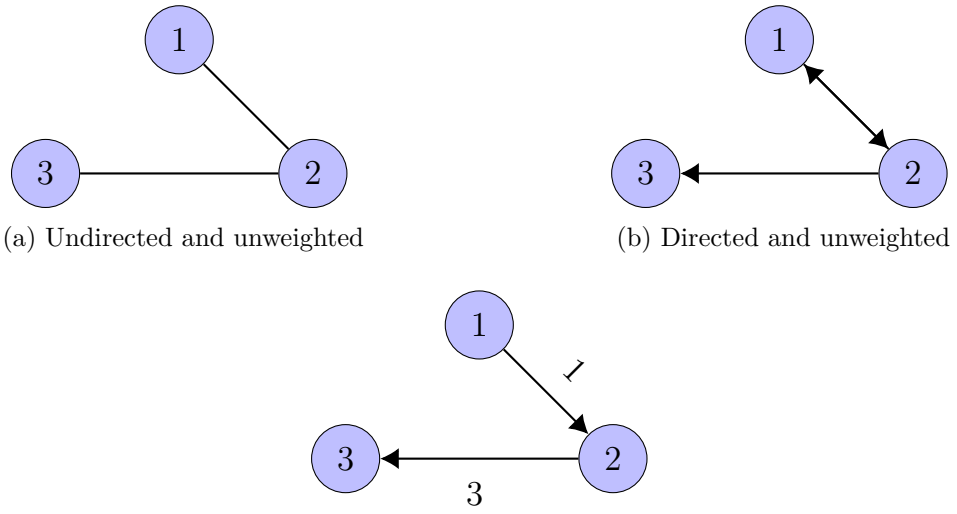
2.1.1 Graph Definition

A *graph* (also called a *network*) G is characterized by a set of *nodes* (V) and a set of *edges* (E), represented as $G = \{V, E\}$. Nodes can correspond to any category of objects that can have a connection among themselves (a number, a name, a city, ...), and an edge stands for a connection between two nodes.

A *directed graph* is a graph where its edges are directed, meaning that each edge represents a unidirectional connection, where for each edge the two nodes have different roles (source node and end node). While in a *weighted graph* edges can have weights associated with them.

The number of nodes is represented as $|V|$, and the number of edges as $|E|$.

We present a few graph examples in figure 2.1.



(a) Undirected and unweighted

(b) Directed and unweighted

(c) Directed and weighted

Figure 2.1: Representation of (a) an undirected and unweighted graph, (b) a directed and unweighted graph, and (c) a directed and weighted graph

2.1.2 Common Metrics

We will now describe some common metrics used to study a graph on its entirety, or a part of it (e.g. a single node). There are metrics used to study individual nodes. Which can then be used to, for example, obtain an average value for the entire graph. With each of those averages being able to represent the graph in a distinct way.

Node Degree The *degree* of a node corresponds to the number of edges connected to it. If the graph is directed then a node has both an *in-degree*, number of edges pointing to it, and an *out-degree*, number of edges pointing from it to another node. We will denote $deg(u)$ as the degree of node u. While directed networks we will use $deg^{in}(u)$ and $deg^{out}(u)$ for in-degree and out-degree values respectively. In which case the degree will correspond to the sum of both in-degree and out-degree.

$$deg(u) = deg^{in}(u) + deg^{out}(u) \tag{2.1}$$

For weighted graphs, the degree of a node is the sum of the weights of the edges connected with that node. Furthermore, if it is also directed, the in-degree (out-degree) of a node is the sum of the weights of the edges that start (end) on it.

The notation used for weighted graphs will be: $deg_w(u)$, $deg^{in}_w(u)$ and $deg^{out}_w(u)$.

In figure 2.2 we illustrate the degree metric by presenting the degree value of each node on an example graph.

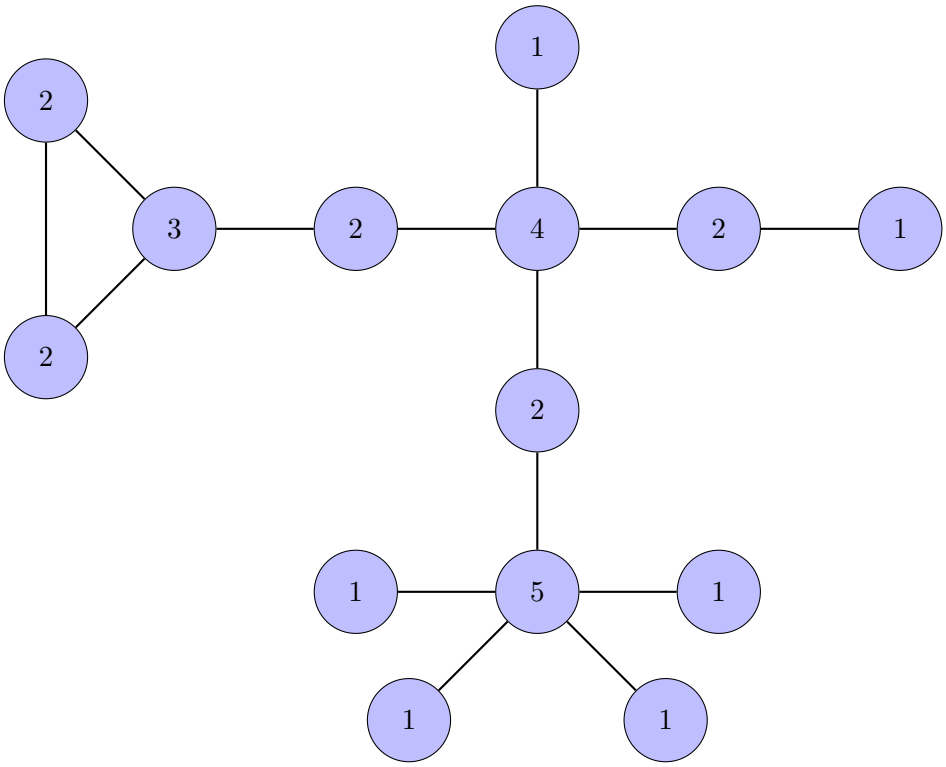


Figure 2.2: Representation of example graph with nodes labelled with their degree values

Average Degree An *average degree* can then be used as a measure of a graph, it is the arithmetic mean of all nodes' degree. We represent it as $\overline{deg}$.
For directed graphs we can then calculate both the *average in-degree* and the *average out-degree*, represented as $\overline{deg^{in}}$ and $\overline{deg^{out}}$ respectively.

$$\overline{deg} = \frac{1}{|V|} \sum_{u \in V} deg(u) \tag{2.2}$$

$$\overline{deg^{in}} = \frac{1}{|V|} \sum_{u \in V} deg^{in}(u) \tag{2.3}$$

$$\overline{deg^{out}} = \frac{1}{|V|} \sum_{u \in V} deg^{out}(u) \tag{2.4}$$

For all three metrics we can also have a weighted version in which the weights are now considered.

$$\overline{deg_w} = \frac{1}{|V|} \sum_{u \in V} deg_w(u) \tag{2.5}$$

$$\overline{deg_w^{in}} = \frac{1}{|V|} \sum_{u \in V} deg_w^{in}(u) \tag{2.6}$$

$$\overline{deg_w^{out}} = \frac{1}{|V|} \sum_{u \in V} deg_w^{out}(u) \tag{2.7}$$

Path Length A *path* between two nodes is a set of nodes where each consecutive pair of nodes is connected by an edge present in the graph. If the graph is directed then the edge must be

directed from the first node in each pair to the second. *Path length* is then the number of edges present in the path, which matches the number of nodes minus 1.

In case of weighted graphs the edge weights are seen as distances and the path length is the sum of the weights of the edges that are part of the path.

The *shortest path length* is simply the length of the path, between two specific nodes, with the lowest number of edges. For weighted graphs, it corresponds to the length of the path with the minimum length, so the minimum total sum of all weights of the edges present in the path.

Betweenness Centrality *Betweenness centrality* of a node v corresponds to the sum of the fractions of the shortest paths that pass on v.

$$bet(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)} \tag{2.8}$$

Where $\sigma(s,t)$ represents the number of the shortest paths between nodes s and t, and $\sigma(s,t|v)$ the number of shortest paths between nodes s and t that pass through node v. In weighted graphs, the shortest paths are calculated with the weights as distances, and we will denote it as $bet_w(u)$.

We present an illustration of the metric in figure 2.3.

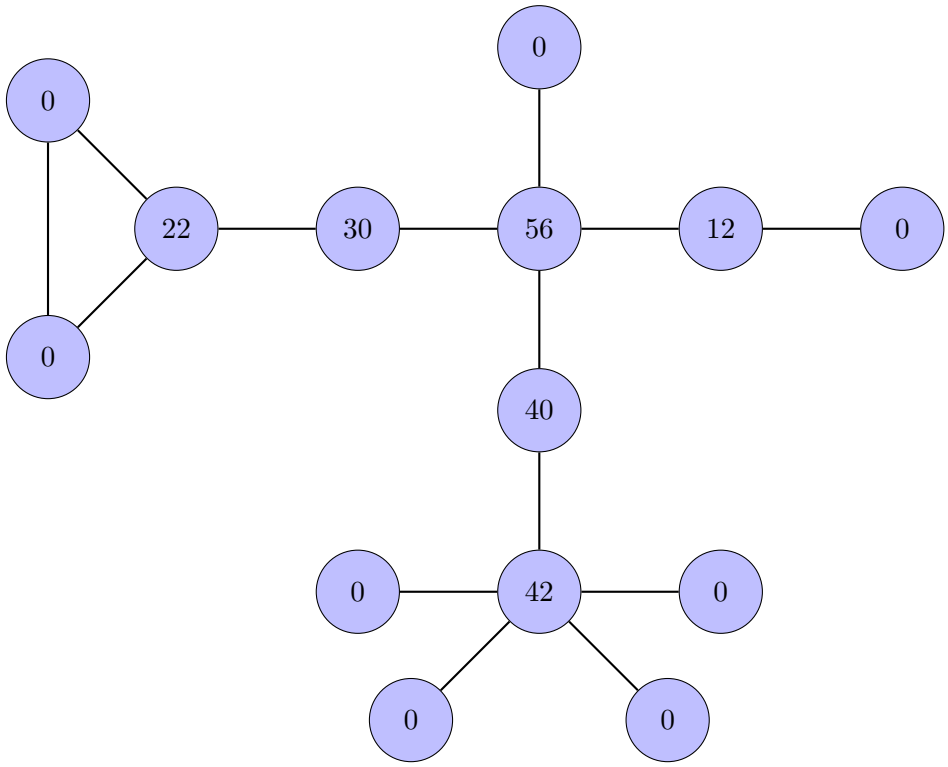


Figure 2.3: Representation of example graph with nodes labelled with their betweenness centrality

Closeness Centrality The metric *closeness centrality* of a node v corresponds to the inverse of the sum of the shortest paths between said node and all other nodes over the number of those reachable nodes.

$$clos(u) = \frac{n-1}{\sum\limits_{u \in V \setminus \{v\}} d(v,u)} \tag{2.9}$$

For weighted graphs, similar to betweenness centrality, the weights are seen as distances when calculating the shortest paths. The weighted version of the metric is then denoted as $clos_w(u)$.

An example of the metric is shown in figure 2.4.

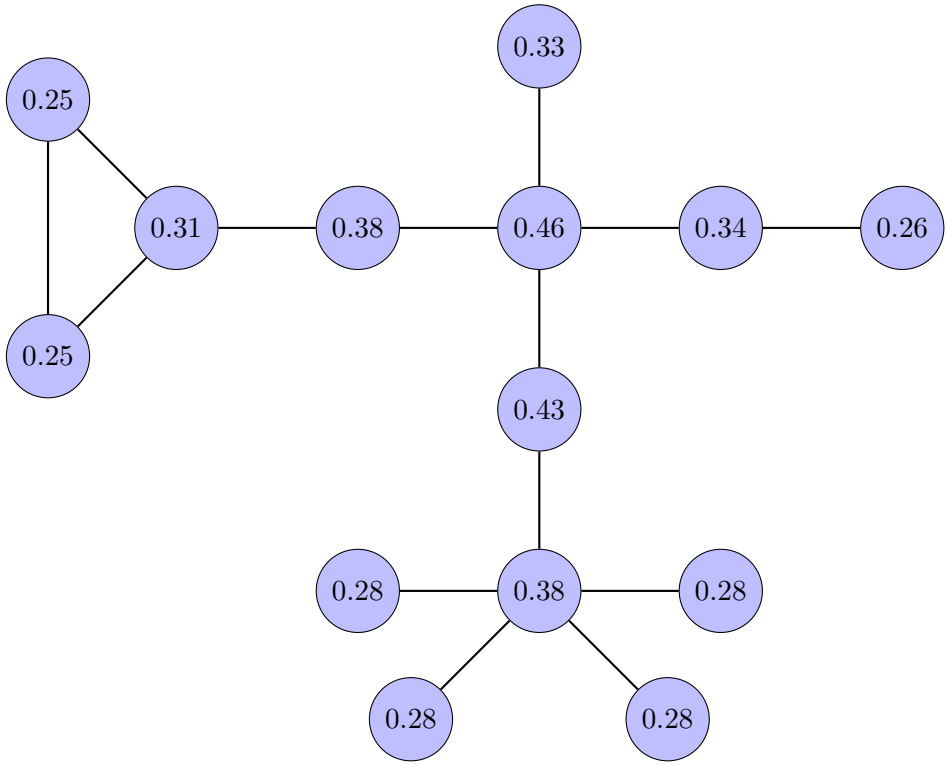


Figure 2.4: Representation of example graph with nodes labelled with their closeness centrality

Clustering Coefficient To measure how clustered, i.e. how connected, the graph is when viewed in small subsets of nodes, we use the *clustering coefficient*. It corresponds to the fraction of triangles (set of three nodes where every pair of nodes is connected) that node v belongs to.

$$clust(v) = \frac{2T(v)}{deg(v) \times (deg(v)-1)} \tag{2.10}$$

$T(v)$ represents the number of triangles containing v, and $\frac{deg(v) \times (deg(v)-1)}{2}$ the number of pairs of nodes. Meaning that the clustering coefficient is the resulting division of the first value by the second.

In figure 2.5 we display a graph example that has each node labelled with its clustering coefficient. In it, we can see that all nodes but the ones composing the triangle on the upper left part, which is the only triangle in the graph, have clustering coefficient 0.

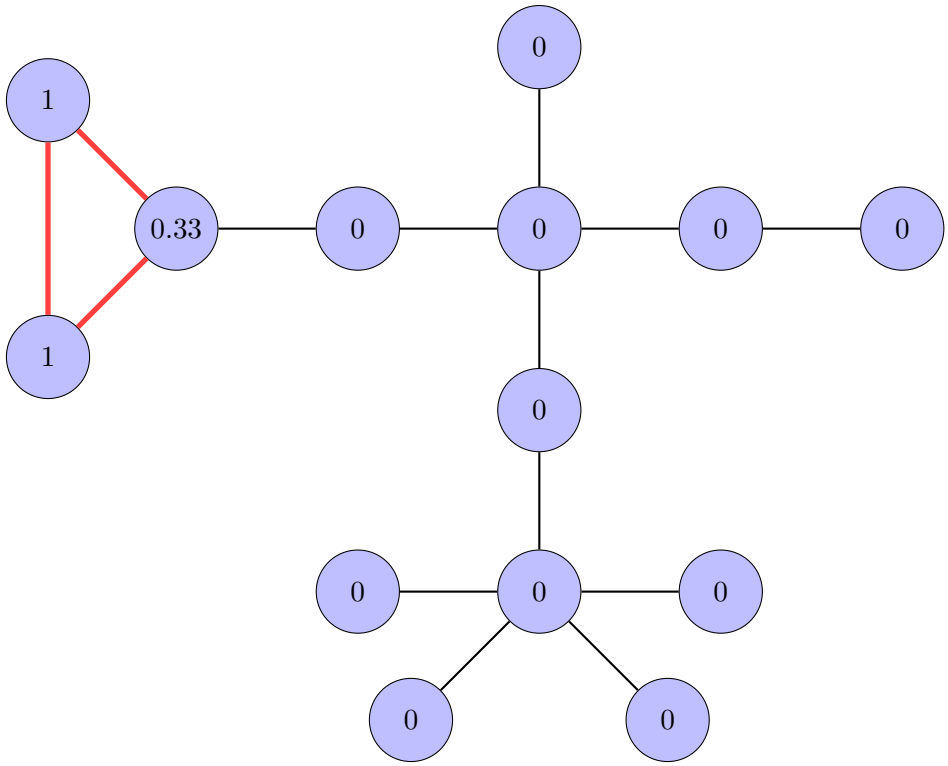


Figure 2.5: Representation of example graph with nodes labelled with their clustering coefficient

Density *Density* measures how many edges there are in relation to the number of nodes. I.e. the fraction of existing edges compared to the total number of possible edges. So for an undirected graph, we have

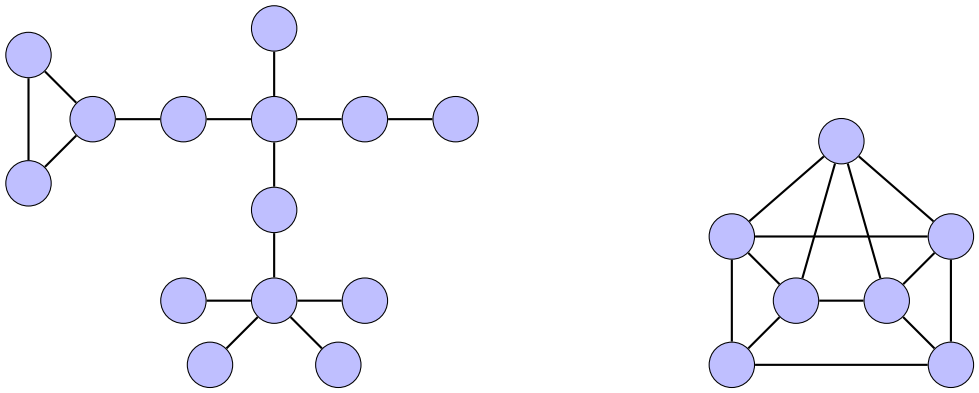
$$\frac{2|E|}{|V| \times (|V| - 1)} \tag{2.11}$$

While for a directed one:

$$\frac{|E|}{|V| \times (|V| - 1)} \tag{2.12}$$

In figure 2.6 we display two graphs, graph (a) has 14 nodes and 14 edges. Therefore, having a density of $\frac{2\times14}{14\times13} \approx 0.154$.

For graph (b) its density value is of $\frac{2\times13}{7\times6} \approx 0.619$. Since it has fewer nodes and more edges, it has a higher density value.



(a) Low density (b) High density

Figure 2.6: Representation of a graph with (a) low density and (b) high density

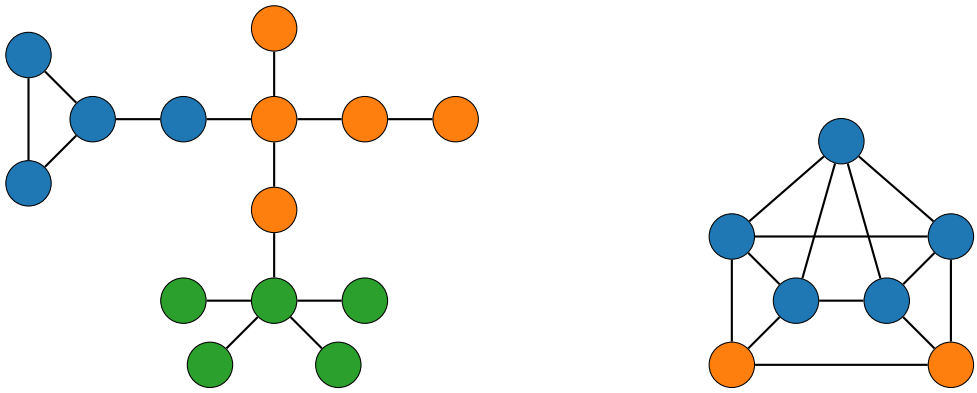
Communities and Modularity *Communities* are subsets of nodes that partition the entire set of nodes.

Modularity is the metric that measures how well divided the communities of a network are. A community partition with high modularity means that the nodes of each community have "many" connections between themselves and there are "few" connections between nodes of different communities.

There are several algorithms to detect the communities of a network. The one used in our study was the Louvain Algorithm [2], through a `NetworkX` implementation. The Louvain Algorithm partitions the network in communities by iteratively improving the modularity value.

It is worth mentioning that the implementation does not work with directed networks, and so we are using it on an undirected version of the networks we have created.

In figure 2.7 we present two examples of graphs divided by communities that were obtained via the Louvain Algorithm. The graph (a) has a modularity value of 0.52, while graph (b) modularity value is 0.05. By looking at both examples we can see that in (a) the nodes are more clearly separated than in (b), having many more edges in between the nodes of the same community than between nodes of different communities.



(a) High modularity (b) Low modularity

Figure 2.7: Representation of graphs divided by communities: (a) with high modularity and (b) with low modularity

2.2 Musical Concepts

A musical score is a common musical notation used to represent music in a way that can be read by people. It does so by matching individual sounds to notes. Where a note is associated with a certain *pitch* and *duration*.

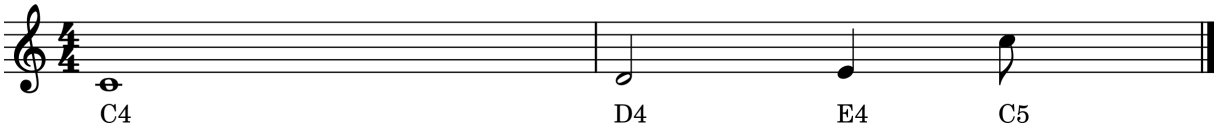


Figure 2.8: Musical score example

In the figure 2.8 we present a very simple musical score with only four notes. The duration of each note is given by the shape of the symbol used, and its pitch by its vertical position on the lines of the staff (the set of 5 horizontal lines).

The pitch of a note corresponds to the sound's frequency. For example, the notes in the score, in order of appearance, match the following frequencies:

- $C4 \rightarrow 261.6256$ Hz
- $D4 \rightarrow 293.6648$ Hz
- $E4 \rightarrow 329.6276$ Hz
- $C5 \rightarrow 523.2511$ Hz

An *octave* is a set of notes where the note with the highest pitch's frequency is double the value of the frequency of the note with the lowest pitch. The notes $C4$ and $C5$ are examples of notes separated by an octave, as C5's frequency is double that of C4's.

As for the sound duration, the first note occurring in the score is called a whole note, the

second a half note, the third a quarter note, and the fourth an eighth note. Half a note lasts half the duration of a whole note, a quarter note lasts half the duration of a half note, and an eighth note lasts half the duration of a quarter note. There are other note durations, and they are always half the duration of the first duration value that is longer than it.

In the example score we can observe that the spacing between the notes is not the same. As the different notes have different durations, they occupy more or less space in the score given their duration, which represents how much time they occupy.

How long a beat lasts in seconds is given by the *tempo*. Tempo is the number of beats per minute, and the same music at a different tempo essentially plays at a different speed or rhythm. It is often stated at the top of the score, represented with a quarter note symbol and the value of beats per minute.

How many beats a whole note is equal to is defined by the *time signature*, the two numbers at the start of the score. The value on the bottom defines how many beats the whole note lasts. So in the example a whole note lasts 4 beats. Which means a half note then lasts 2, a quarter note lasts 1, and so on.

In 2.9 we present the first row of a score of an actual music. There are of course many other concepts needed to be able to fully understand a musical score, but the ones mentioned are the most relevant ones for our purposes.

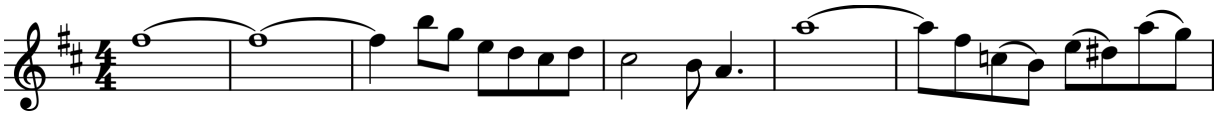


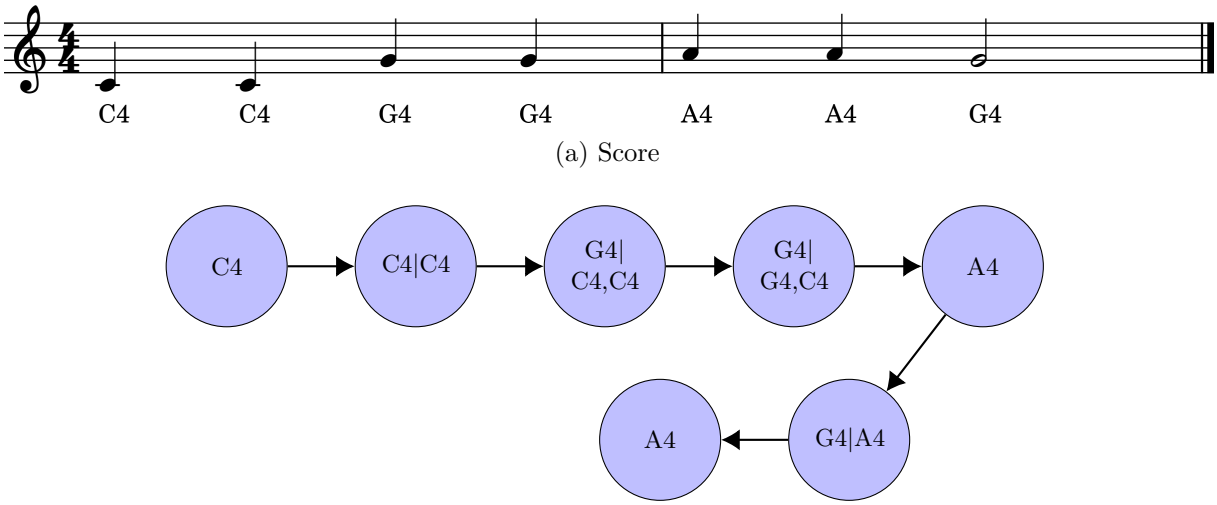
Figure 2.9: An excerpt of Air on a G String (Bach)

2.3 Related Work

There is already some work done to study music through complex networks. In one such study [7] guitar solos as the target of study, a similar feature set is used but with a focus on the length of the solos as well.

One approach used *higher order networks* [25] to study music, having each node able to represent not just a single note but sequences of notes according to dependency [23]. An example can be seen in figure 2.10, where the score (a) would be converted to the graph (b). The rules are extracted choosing the dependencies with high confidence and sufficient support.

At one study [21] songs were split into time intervals with equal length. From that division a *pitch transition network* would be created, in which each node represented a set of notes that are played in the same interval. Each node then linked to nodes containing a set of notes that happen in the previous or following interval, the edge weight reflecting the frequency of that



(a) Score

(b) Mapped graph

Figure 2.10: Higher order network example

sequence of sets. Those networks would then be used to compare different songs, using degree distribution, as well as common metrics such as average path length and clustering coefficient.

Yet another approach has been to study the harmonic side of music, i.e. notes being played simultaneously [9]. In it, a node would consist of nodes that are played at the same time as represented in the musical score. Edges would then be created as in most other studies, linking nodes with a set of simultaneous playing notes whenever these sets of nodes occur one after the other.

Chapter 3

Songs to Networks

In this chapter we present what we used to study music, explaining what the MIDI format is and how it can be useful, particularly for our study. As well as discuss how we mapped songs into networks, going through each phase and showing each song in various visual formats.

3.1 Data

There are two types of music representation: **symbolic** and **audio**.

In symbolic representation (its most common format being `MIDI`) the song is recorded via music notation, which allows us to look directly at the musical aspects of the song as it was composed.

Whereas in audio only the audio waves are recorded (in formats such as `MP3` or `FLAC`), and so we can only analyse the final product which lacks the description used when creating a song.

So while audio might be better to replay and listen to, the symbolic representation allows for a different analysis of the song, with a strong focus on its elemental musical concepts, such as notes' pitch and duration. So with it, we can attempt to better grasp how those elements affects the music in its whole structure.

In this study we will be analysing songs resorting to symbolic representation through MIDI files, as there is easy access, and as mentioned before it will provide us a means to look at a song through its music elements. Our datasets are all subsets of The Largest MIDI Collection on the Internet [1] and Musescore.com [2] (the latter specifically for classical music).

[1] `https://www.reddit.com/r/WeAreTheMusicMakers/comments/3ajwe4/the_largest_midi_collection_on_the_internet/`, 2022

[2] `https://musescore.com/`, 2022

3.1.1 MIDI

The MIDI format is ideal for our purposes, since through it, we can analyse the song note by note. Each MIDI file can have one or more tracks. When a song is split into several tracks, each track corresponds to only one instrument. But one instrument can be represented in more than one track. For example when playing piano, often each hand plays a different part of the song, which is distinctively separated into two staffs on a musical score, as shown in the example of figure 3.1. MIDI files save each part represented by a staff as an individual track.

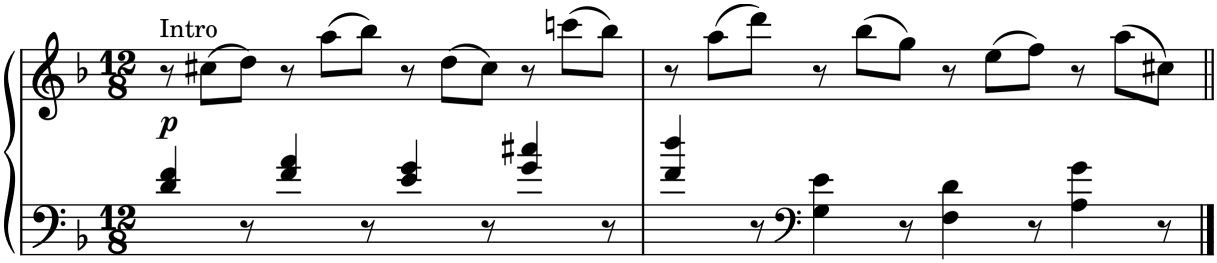


Figure 3.1: An excerpt of Lacrimosa from the Requiem in D minor (Mozart)

However, there is not an obligatory standard way of recording the midis, at least among the dataset we worked with. Where sometimes, the same instrument can have multiple tracks.

A track in a MIDI file is essentially an ordered list of messages, a message being either a `Message` or a `MetaMessage`, where each message makes a change to the track on a given time. The messages that we will focus on are the `Messages` of type `note_on` and `note_off`, which serve mainly to determine when an instance of a note starts or ends. Each of those messages specify a note, octave, and time. Note will hereby be used to refer to the pair: note and octave. As each pair is represented in MIDI by a note number from 1 to 127, that is what we used to identify the note.

Higher note numbers will represent higher octaves. The note numbers 0 to 11 correspond to the notes in the first octave, 12 to 23 to the notes in the second, and so forth. We show in table 3.1 the full list of notes matching their usual musical notation with the note numbers used in MIDI. One note number per pair (note and octave).

In the example message showed in figure 3.2, the message being of the type `note_on` is then starting a note. Each message of either type has as attributes: `channel`, `note`, `velocity`, and `time`.

```
Message('note_on', channel=0, note=60, velocity=80, time=20)
```

Figure 3.2: MIDI message

Each `channel` represents the information of a specific instrument, and unlike the tracks

Octave	Note Number											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Table 3.1: Representation of notes in their usual notation mapped to MIDI's note numbers

each instrument is only represented by a single `channel`.

The value for `note` is the note number that represents the notes in MIDI.

`velocity` essentially measures how "hard" a note is played. For instance how hard a piano key is played.

The `time` attribute, which exists in all messages, indicates how much time after the previous message the current one will occur.

`Message` can also be used to define the instrument to play the notes, called `program` in MIDI. Besides `Message` there is also another type called `MetaMessage`. This serves for example to: end a track, marking the time it ends; set the value for `tempo` for the rest of the track's duration; define a name for the track.

The `time` attribute of each message is delta time in *ticks*. So the number of ticks the current message will occur after the previous message. Where the duration in microseconds of each tick depends on the tempo, and the number of ticks per beat, defined for the song.

The value for tempo specifies the duration of each beat. Unlike the standard music concept, in MIDI files tempo is given as microseconds per beat, instead of beats per minute. We will refer to MIDI's parameter for tempo as `tempo`.

Each beat contains a specific number of ticks. That number is given by the parameter `ticks_per_beat`.

The parameter `ticks_per_beat` is fixed for the whole song, defined as a general attribute of the MIDI file and not via a `Message`, while the `tempo` can be changed at any time throughout the song via a `MetaMessage`.

An example of the mentioned units is given in the figure 3.3 (taken from `Mido`'s page).

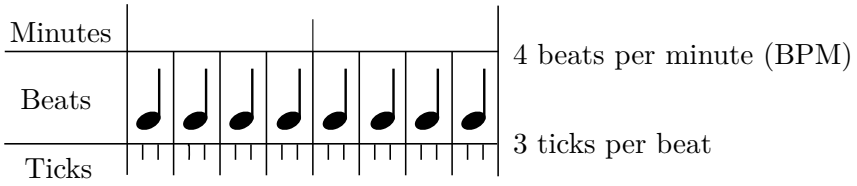


Figure 3.3: Representation of MIDI time units

3.1.2 Processing

For every song we analyse only one track. In our analysis, we manually picked a track for each song. Ideally the track would be chosen automatically, but we have not found a suitable method to make such choice.

A note can then be analysed by looking at the two messages that indicate the note's starting and ending time. Which is done by looking at the consecutive messages with the same note number.

After having chosen a track, we obtain a list of ordered notes, by going through the aforementioned messages `note_on` and `note_off`, creating pairs of messages that link a note's starting and ending time. The notes each being represented by their MIDI note number. Since notes can overlap (one starting before a previous one having finished) what are consecutive notes is not straight forward. For our mapping we chose to order the notes by their *starting time*.

In figure 3.4 we present only the `note_on` and `note_off` messages of a MIDI track. The pairing of the messages is represented by colour and number. If two messages are paired, then one starts the note and the other ends it.

In the example there are four messages which have the note code 45. Two of them start a note with that note number, and another two end it. As mentioned before we pair them simply by the order in which they appear. Since the notes are ordered by their starting time, the resulting order would then be as follows: 45 → 60 → 45.

```
1 - Message('note_on', channel=0, note=45, velocity=80, time=20)
2 - Message('note_on', channel=0, note=60, velocity=80, time=20)
1 - Message('note_off', channel=0, note=45, velocity=80, time=20)
3 - Message('note_on', channel=0, note=45, velocity=80, time=20)
2 - Message('note_off', channel=0, note=60, velocity=80, time=20)
3 - Message('note_off', channel=0, note=45, velocity=80, time=20)
```

Figure 3.4: MIDI message pairing

3.2 Mapping

Given the ordered notes list, we create from it a directed graph to map the track.

Each unique note appearing in the list would correspond to a node. With a single node per note (note code), i.e. multiple notes of the same note code would not give origin to different nodes. For each ordered pair of notes we then create a directed edge between the corresponding nodes. The edge having as weight the number of times that the pair occurs in the song, as to differentiate how meaningful an edge (pair of notes) is to a song.

Rests Another thing we considered as nodes were *rests*. MIDI files do not need to explicitly define a rest, as it occurs naturally whenever there are no notes playing for a certain time. As the way we were creating the graph did not include those rests we decided to add them. Adding them enriched the graph as a means to study a song, since silence is as much of a choice as any particular note, as it is in fact choosing not to play any note. We considered a rest whenever there was an interval of time between notes that was larger than a threshold. The threshold corresponding to the minimum duration of all notes of the song (at least 1 tick). If the interval of time is below the maximum duration of all notes then they are considered a *short rest* associating it with the note number 128, if not we consider them as a *long rest* with the note number 129.

We considered at most one rest (short or long) between two consecutive notes. So there will never be an edge between the nodes corresponding to the rests.

After some experimenting we quickly realized that with these default values the mapped graphs of the songs would be centred on those rest nodes. Having many more rests than each unique note present. To correct this aspect we added three parameters, for fine-tuning, to obtain representations that still accounted for rests, but in a way that it did not overshadow information from the actual notes. Instead of simply using the minimum duration as the lower threshold, we used it multiplied by a parameter (`SHORT_REST_THRESHOLD_MULTIPLIER`).
Similarly, we added a multiplier (`LONG_REST_THRESHOLD_MULTIPLIER`) to the maximum duration, to define the border between short and long rest.

Since many rests would happen in between too long periods, which would be explained by music being played in other tracks at the time, we also added a maximum value for rests. This parameter would also be a multiplier (`REST_MAX_MULTIPLIER`) of the longest note duration.

For the datasets and analysis mentioned later on in this study, we used the values:

- `SHORT_REST_THRESHOLD_MULTIPLIER` = 200
- `LONG_REST_THRESHOLD_MULTIPLIER` = 1
- `REST_MAX_MULTIPLIER` = 5

3.2.1 Mapping stages

We illustrate our mapping by going through the different stages. We process a MIDI file via the Python package `Mido`, and we obtain the list of messages as shown in figure 3.5. As previously mentioned we will only focus on the messages that start or end a note, which are the ones of type `note_on` and `note_off`. In this example we will only need to pay attention to track 1, as track 0 does not even start or end any note and only sets some parameters (e.g. `tempo`) for the MIDI file.

```
0: MidiTrack([
  MetaMessage('time_signature', numerator=4, denominator=4,
      clocks_per_click=24, notated_32nd_notes_per_beat=8, time=0),
  MetaMessage('set_tempo', tempo=500000, time=0),
  MetaMessage('end_of_track', time=0)])

1: MidiTrack([
  MetaMessage('track_name', name='Ragtime Piano', time=0),
  Message('program_change', channel=0, program=3, time=0),
  Message('note_on', channel=0, note=60, velocity=80, time=20),
  Message('note_off', channel=0, note=60, velocity=80, time=30),
  Message('note_on', channel=0, note=62, velocity=80, time=20),
  Message('note_off', channel=0, note=62, velocity=80, time=60),
  Message('note_on', channel=0, note=60, velocity=80, time=20),
  Message('note_off', channel=0, note=60, velocity=80, time=30),
  Message('note_on', channel=0, note=64, velocity=80, time=20),
  Message('note_off', channel=0, note=64, velocity=80, time=30),
  Message('note_on', channel=0, note=64, velocity=80, time=20),
  Message('note_off', channel=0, note=64, velocity=80, time=30),
  Message('note_on', channel=0, note=60, velocity=80, time=20),
  Message('note_off', channel=0, note=60, velocity=80, time=30),
  MetaMessage('end_of_track', time=0)])
```

Figure 3.5: MIDI message list

From the track 1 we present it as a musical score in figure 3.6. While MIDI records information as a note's velocity, which in a piano, for example, represents how hard or soft a key is played, the score does not. Furthermore, MIDI can hold precise values for duration of a note, and time in between different notes' starts and ends, that can be much shorter than the ones on a score. Thus, the score holds less information than a MIDI file. Which is to be expected as a score is to be read by a human whereas a MIDI file by a MIDI player.

The notes in the example MIDI file are mapped from their note numbers to the usual score representation as:

- $60 \rightarrow C4$
- $62 \rightarrow D4$
- $64 \rightarrow E4$

Having processed the messages from a chosen track, we will then map those notes to a directed

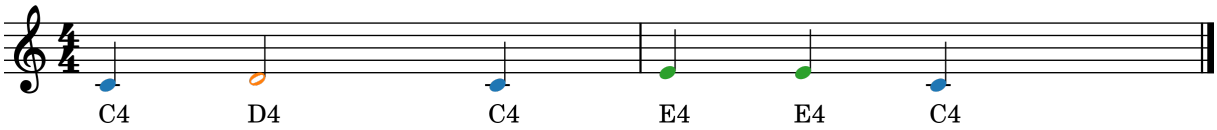


Figure 3.6: Musical score representation of the MIDI message list: each note being now represented as a single element as opposed to a pair of messages

graph. What we look to map the notes is the order in which they appear. But as mentioned, the way our mapping works, to create the edges we pay attention solely to the notes' starting time. To visualize the times of each note, we present the notes of track 1 in a timeline in figure 3.7. We note that all the notes have the same duration except for the note with code 62 (represented as D4 in the score), which lasts twice as long as the others. That information, the duration of each note, is kept in the timeline figure.

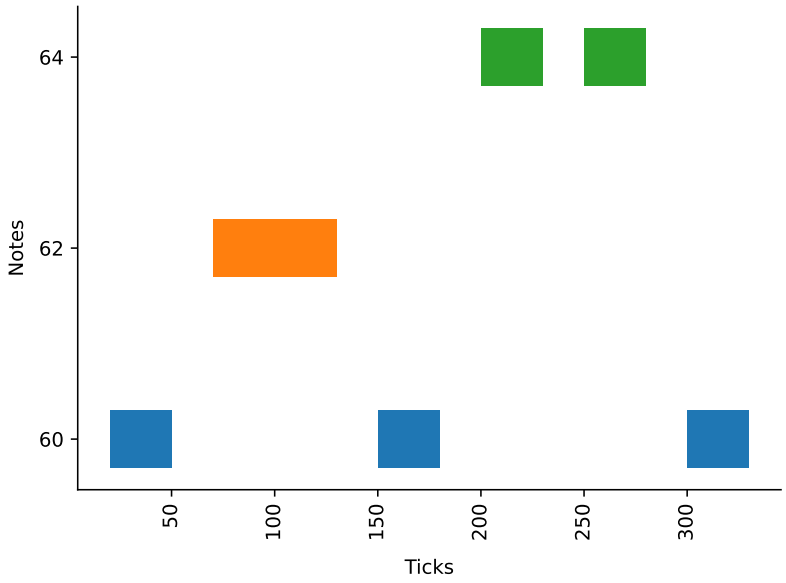


Figure 3.7: MIDI timeline representation of each note across time, representing their starting and ending times, as well as their duration

Having the notes ordered by their starting time, we then create each occurring note as a node, and create a directed edge between two consecutive notes. The edge having as weight the number of times that ordered pair occurs. The resulting directed graph is presented in figure 3.8. Our mapping does not preserve the duration aspect of each note, and so the presented graph ignores this information.

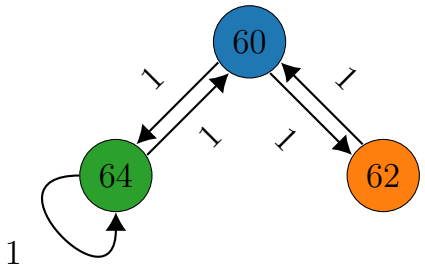


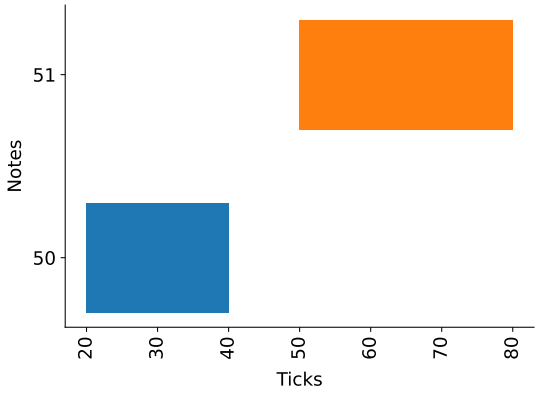
Figure 3.8: Representation of the mapped graph obtained from the notes of the MIDI message list

3.2.2 Overlap

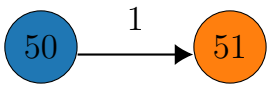
The examples shown so far had at all instances at most one note playing. But even within a single track, we can have the case where there are several notes playing at the same time.

With our mapping an edge is created whenever a note follows another. That order being given by the starting times of those notes.

A sequence of notes as shown in figure 3.9(a) has the note 50 start before the note 51, so in the mapped graph the node corresponding to note 50 has an edge to the node representing note 51. The mapped graph is presented in figure 3.9(b). In this case the notes are clearly separated. The second note starting after the first ends.



(a) Timeline



(b) Mapped graph

Figure 3.9: Overlap example 1

Since the starting time alone defines the order of the notes, and thus the direction of the edges, even if notes overlap the mapping can remain the same. In figure 3.10 the note 51 overlaps with note 50, i.e. there is a period where they are both playing. But because the order of the starting time is the same, the mapped graph is the same as in example 1.

In example 3, shown in figure 3.11, the note 51 starts and ends while the note 50 is playing. But like in example 2, despite the notes overlapping, the resulting graph is the same as the one of example 1.

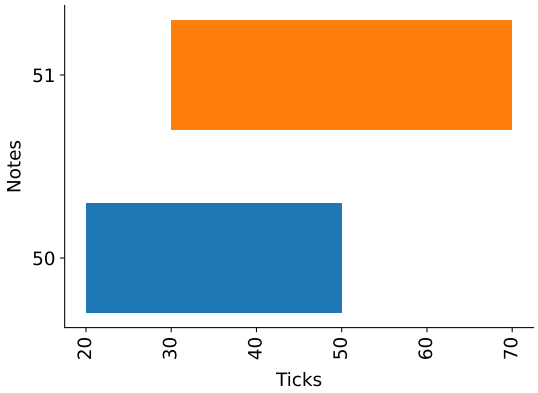


Figure 3.10: Overlap example 2

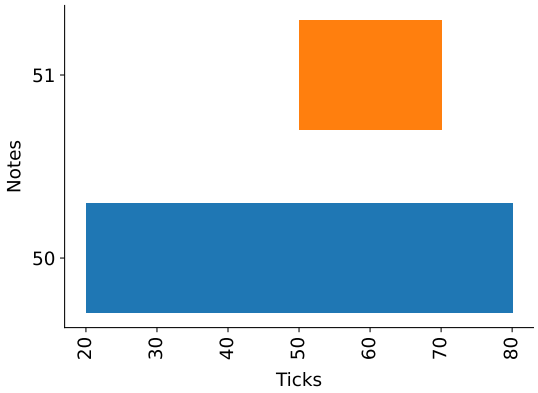
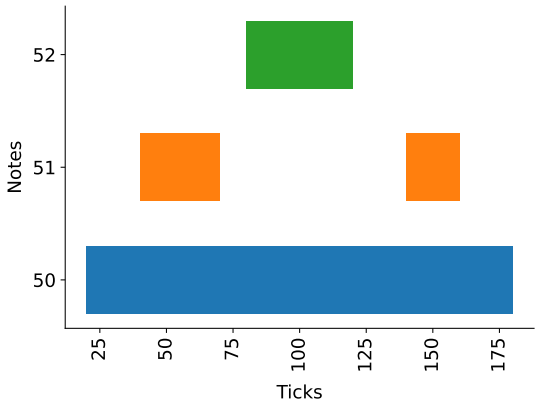


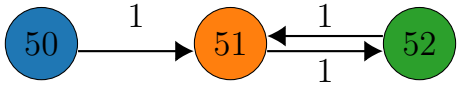
Figure 3.11: Overlap example 3

With our mapping only one connection will be counted for each note instance. Unless if it is the final note to be played, in which case it will not create an edge or increase an existing one's weight.

In figure 3.12, even though the note 50 is played throughout the entire time, it is only mapped as the first note. Having only a connection to note 51, which was the first note to start after it. Despite the fact that while note 50 played, the note 51 had been played twice and note 52 once.



(a) Timeline



(b) Mapped graph

Figure 3.12: Overlap example 4

Chapter 4

Analysis

Our goal was to understand what music characteristics we could identify when resorting to features obtained from the network that mapped the song. We did this by analysing sets of songs, through a chosen feature set.

4.1 Features

Since most metrics evaluate at a node level [4], we obtained the arithmetic average of the metric values for all nodes present in the network to create a feature that could be used to represent the whole network. There is an expected loss of information in doing so, but it allows us to compare networks more easily.

We have already mentioned average degree (as well as average in-degree, and average out-degree), we calculate the average of other metrics in the same manner.

$$\overline{bet_w} = \frac{1}{|V|} \sum_{u \in V} bet_w(u) \tag{4.1}$$

$$\overline{clos_w} = \frac{1}{|V|} \sum_{u \in V} clos_w(u) \tag{4.2}$$

$$\overline{clust_w} = \frac{1}{|V|} \sum_{u \in V} clust_w(u) \tag{4.3}$$

When comparing the features of each graph, among the entire set of graphs, because different musics can have different lengths, the absolute values could vary enough that even if two songs are alike (with respect to the features we use) it would not be noticed by our study. Thus, we chose to normalize the metrics' values to the interval $[0, 1]$. The metrics that are not normalized by default, are normalized by comparing to the max and min values of that feature across all songs of the set.

We will denote the normalized version of a $metric(u)$ as $metric^n(u)$.

We normalized the average weighted in-degree, the average weighted betweenness centrality, and the average shortest path length as follows. For each dataset we took the minimum (min) and maximum (max) values of each metric, and linearly mapped the values from the interval $[min, max]$ to the interval $[0, 1]$, resorting to the simple mapping:

$$\begin{aligned} f\colon [min, max] &\to [0, 1] \\ x &\mapsto \frac{x - min}{max - min} \end{aligned}$$

As for density, modularity, closeness centrality, and clustering coefficient, by definition these metrics already output values in the interval $[0, 1]$.

`NetworkX` was used to calculate the node's degree, betweenness centrality, closeness centrality, clustering coefficient and average shortest path length. As well as the graph wide metrics: modularity and density.

The features we selected for our analysis were then:

- Average Weighted In-degree
- Average Weighted Betweenness Centrality
- Average Weighted Closeness Centrality
- Average Clustering Coefficient
- Average Shortest (Weighted) Path Length
- Density
- Modularity

Other features could have been chosen. We decided to stick to "core" complex networks features, hoping to capture similarly "basic" musical aspects. Sets of features including most of the ones we chose have been used in other complex network centred music studies [7, 16].

Our intent when choosing the features was for them not to be redundant. To have each capture some different aspect of the song, even if it does not translate by itself as a characteristic that we can measure. As it can be that we will have to use two or more to be able to do so for some musical aspects.

4.2 Synthetic Dataset

To understand the characteristics that could be studied with this feature set we designed, and created, a few synthetic MIDI files. The goal was to first compare the feature values of examples that were easier to understand than actual songs.

4.2.1 Designing The Dataset

Most of the examples of this dataset were created to present simple patterns to evaluate how our feature set worked in capturing musics' characteristics. Despite none of the examples being music as perceived by us, they still represent sound patterns that could exist in songs, even if in a less "perfect" state. The ones that do not contain a simple pattern are created fully randomly (or with a simple restriction).

We present a visualization of each, a time series representation, in figure 4.1. Showing each note by their starting time in ticks, and just like in our mapping we ignore their duration. The notes scale goes from 0 to 129 (the standard MIDI note numbers of 0 to 127, and the short and long rests we have added as, respectively, 128 and 129).

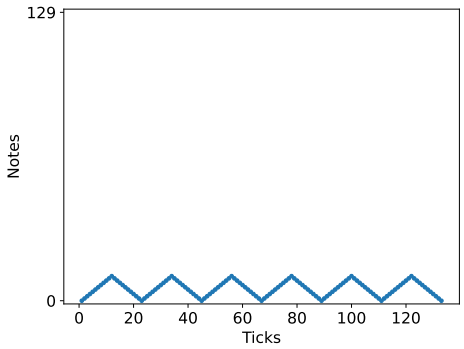
In this Synthetic Dataset, we present 8 examples:

- `peak_fixed_octave` - We fixed the first octave and made a sequence of equal peaks, where a peak is a sequence of notes where each note is a note code above the previous one until it reaches the highest chosen note, then it will descend linearly in the same manner, each note being exactly a note code below the previous one.
- `peaks_small_large_constant` - Similar to `peak_fixed_octave` but with smaller peaks in between each of the larger ones.
- `peaks_small_large_rising` - A version of `peaks_small_large_constant` that is not constant, in the sense that it as an upwards tendency. A repetition of a small peak followed by a larger one, but in this case the larger one does not descend as much as it had risen. Which results in the aforementioned upward tendency.
- `random_fixed_octave` - Each note is obtained by generating a random note code in the interval [0,11] which represents the first octave.
- `random_fully` - A fully random example where each note is obtained as a randomly generated number from one of 128 possible note codes (we excluded the rests as they can not be represented as notes in the MIDI format).
- `repeat_aabb_up` - We repeated each note of the first octave five times, starting with the smaller note code 0 and going up one by one.
- `straight_fixed_octave_up` - In this example we start with the note code 0 and go one by one until we reach the final note of the first octave. We then repeat this sequence 6 times.
- `straight_rising_octave_up` - Each note is obtained as one higher of the previous one, starting with the note code 0.

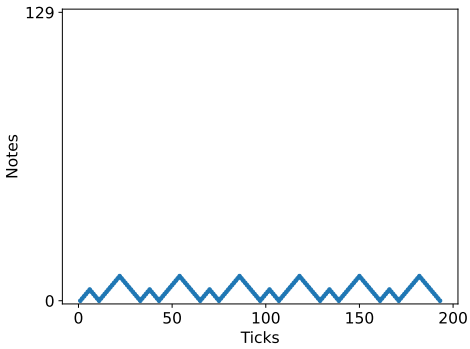
The number of notes on each example is not always the same, as can be seen by the number of ticks on each example's time series representation.

We also present the graph, resulting from our mapping, of each of those synthetic examples in figure 4.2. For each graph the node size is relative to its in-degree, the smallest size matching the lowest in-degree value of the graph, and the largest size the highest. Likewise, the node

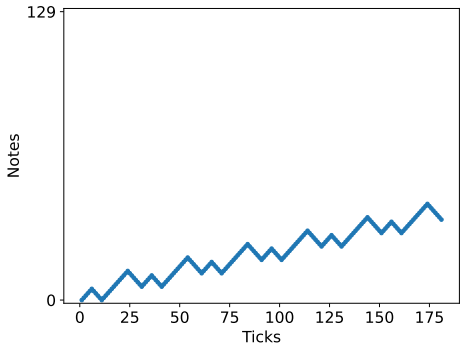
colour is relative to its betweenness centrality value, and the edge colour is relative to its weight (darker colours meaning higher value).



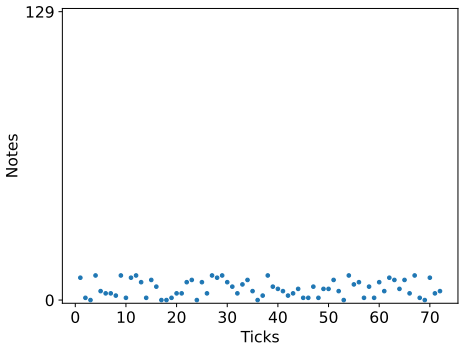
(a) `peak_fixed_octave`



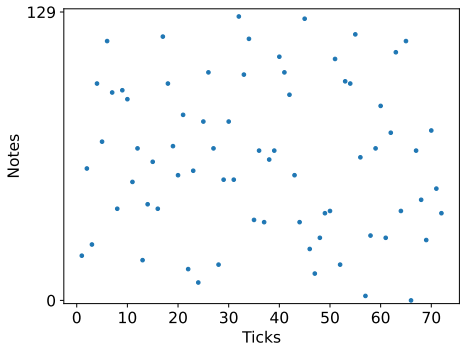
(b) `peaks_small_large_constant`



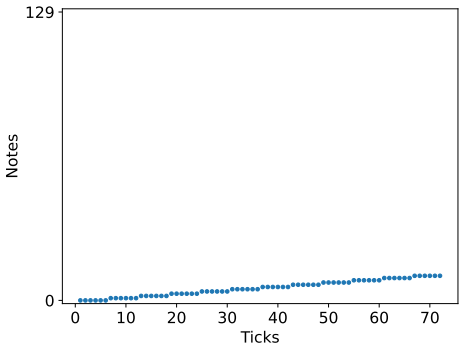
(c) `peaks_small_large_rising`



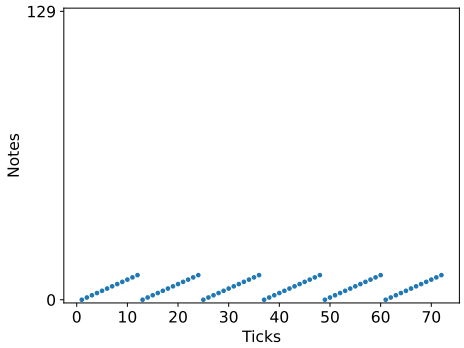
(d) `random_fixed_octave`



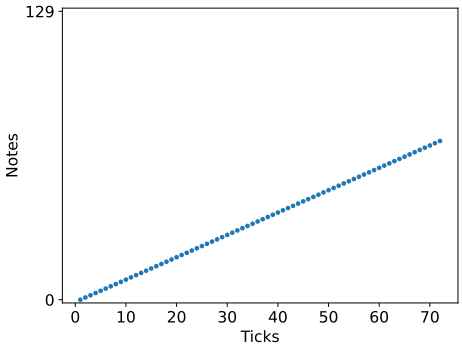
(e) `random_fully`



(f) `repeat_aabb_up`

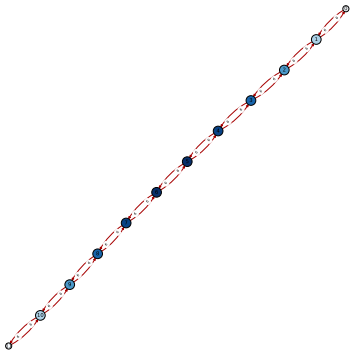


(g) `straight_fixed_octave_up`

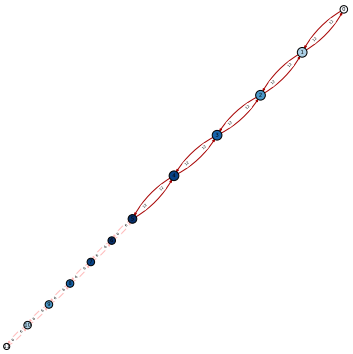


(h) `straight_rising_octave_up`

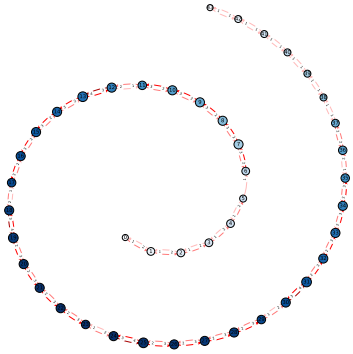
Figure 4.1: Synthetic dataset time series representations



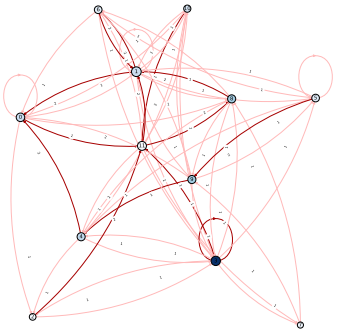
(a) `peak_fixed_octave`



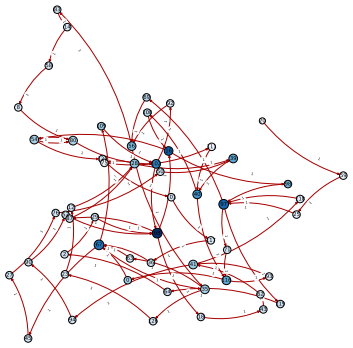
(b) `peaks_small_large_constant`



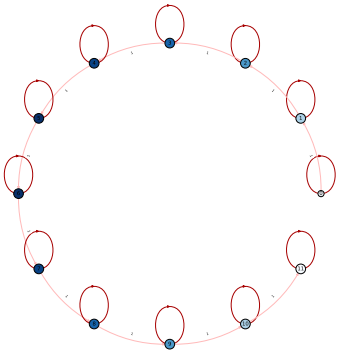
(c) `peaks_small_large_rising`



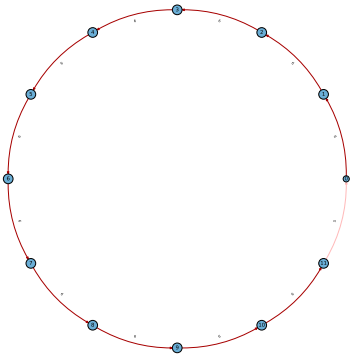
(d) `random_fixed_octave`



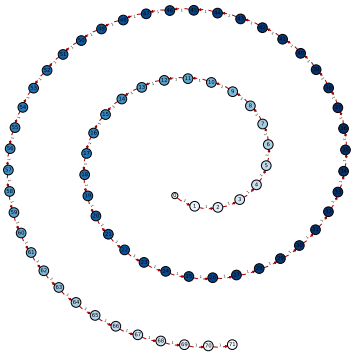
(e) `random_fully`



(f) `repeat_aabb_up`



(g) `straight_fixed_octave_up`



(h) `straight_rising_octave_up`

Figure 4.2: Synthetic dataset graphs with node size relative to in-degree, node colour relative to betweenness centrality, and edge colour relative to weight (darker colours meaning higher value)

4.2.2 Experimental Results

We present the values of the mentioned feature set of each of the synthetic examples in table 4.1. Besides the mentioned features we have also added two columns, under grey coloured headers, containing values that are pertinent to understanding the feature set values: number of nodes ($\#Nodes$) which correspond to the number of unique notes and can be seen in the vertical axis of the time series representation; and number of notes ($\#Notes$) which is the total number of notes occurring in each song's track, each note is represented as a dot on the time series representation of each song.

Song	Avg. W. In-Degree	Avg. W. Betweenness Cent.	Avg. W. Closeness Cent.	Avg. Clustering Coef.	Avg. Shortest W. Path Length	Density	Modularity	#Nodes	#Notes
peak_fixed_octave	11.000	36.667	0.040	0.000	26.000	0.167	0.483	12	133
peaks_small_large_constant	16.000	36.667	0.028	0.000	37.364	0.167	0.479	12	193
peaks_small_large_rising	4.091	493.182	0.031	0.000	29.965	0.045	0.903	44	181
random_fixed_octave	5.917	8.108	0.546	0.438	1.886	0.432	0.136	12	72
random_fully	1.291	308.345	0.138	0.011	6.657	0.024	0.577	55	72
repeat_aabb_up	5.917	18.333	0.135	0.000	2.167	0.174	0.764	12	72
straight_fixed_octave_up	5.917	55.000	0.028	0.000	35.500	0.091	0.427	12	72
straight_rising_octave_up	0.986	828.333	0.026	0.000	12.167	0.014	0.771	72	72

Table 4.1: Synthetic dataset features

Average Weighted In-degree Being weighted the average in-degree is essentially the proportion of notes (related with the total sum of edge weights) and occurring unique notes. Which is why `peaks_small_large_constant` has the highest value, since the occurring notes are all restricted to the first octave, and it is the example with the highest number of notes. `straight_rising_octave_up` being on the opposite side of the spectrum with the highest number of unique notes of the whole dataset, but each note occurring only once.

We can see that it is not just about the number of notes, since `peak_fixed_octave` clearly has fewer notes (as seen also by the number of ticks) than `peaks_small_large_rising` but has a much higher average weighted in-degree. This happens because the notes from `peak_fixed_octave` are all on the first octave, whereas `peaks_small_large_rising` does not have that restriction and so has a much higher number of unique notes. A song being more condensed on a lower number of unique notes, will have a higher average weighted in-degree.

Average Weighted Betweenness Centrality `straight_rising_octave_up` is a single one direction chain. Which makes it so in this example there is at most one directed path between two notes. Because it is a chain, all nodes "before" a specific node will need to traverse it to reach all the nodes "after" it. The number of paths that traverse through a node will then be the product of the number of nodes "before" it, with the number of nodes "after" it. Therefore, the closer to the middle of the chain a node is, the higher its (weighted) betweenness centrality. As can be seen very starkly by looking at its graph.

The `random_fixed_octave` example has a low number of unique notes (being only notes on the first octave) and since all notes were randomly generated is the example the furthest away from a chain. Each node having often many more than just one edge pointing to it.

The other random example `random_fully` has a much higher average weighted betweenness centrality. Since even though each note is generated randomly, the notes are no longer restricted to the first octave. Given the way our mapping works we always establish a connection between two consecutive notes, which makes it likely to result in something resembling a long chain, even if some nodes do end up having more than one edge pointing to them.

Average Weighted Closeness Centrality For the same reasons as mentioned above, `random_fixed_octave` has the highest average weighted closeness centrality. It has very few nodes, and many distinct edge connections.

Similarly, `straight_rising_octave_up` has a very low value because there are many nodes and at most one way to reach each node, often passing through several nodes.

This does not mean however that the metric is redundant. We can see for example that although `straight_fixed_octave_up` and `straight_rising_octave_up` have nearly the same average weighted closeness centrality the latter has a much higher average weighted betweenness centrality, since it is a one way chain with many more nodes than the first.

Average Clustering Coefficient Since most of these examples are chains or near chains, their nodes have very little clustering coefficient. Except for the `random_fixed_octave` that has very few nodes but a high diversification. Having a very high density value, which ends up resulting in many triangles (sets of three nodes where each pair of nodes has a connection).

Average Shortest (Weighted) Path Length Both `peak_fixed_octave` and `peaks_small_large_constant` have high values because all existing edges have high weight, since both examples were created by repeating a certain patten a number of times. Even though `peaks_small_large_rising`'s edges do not have such high weights, because it is also a chain, it has more nodes there are more shortest paths that need to go through more nodes than in the two other examples.

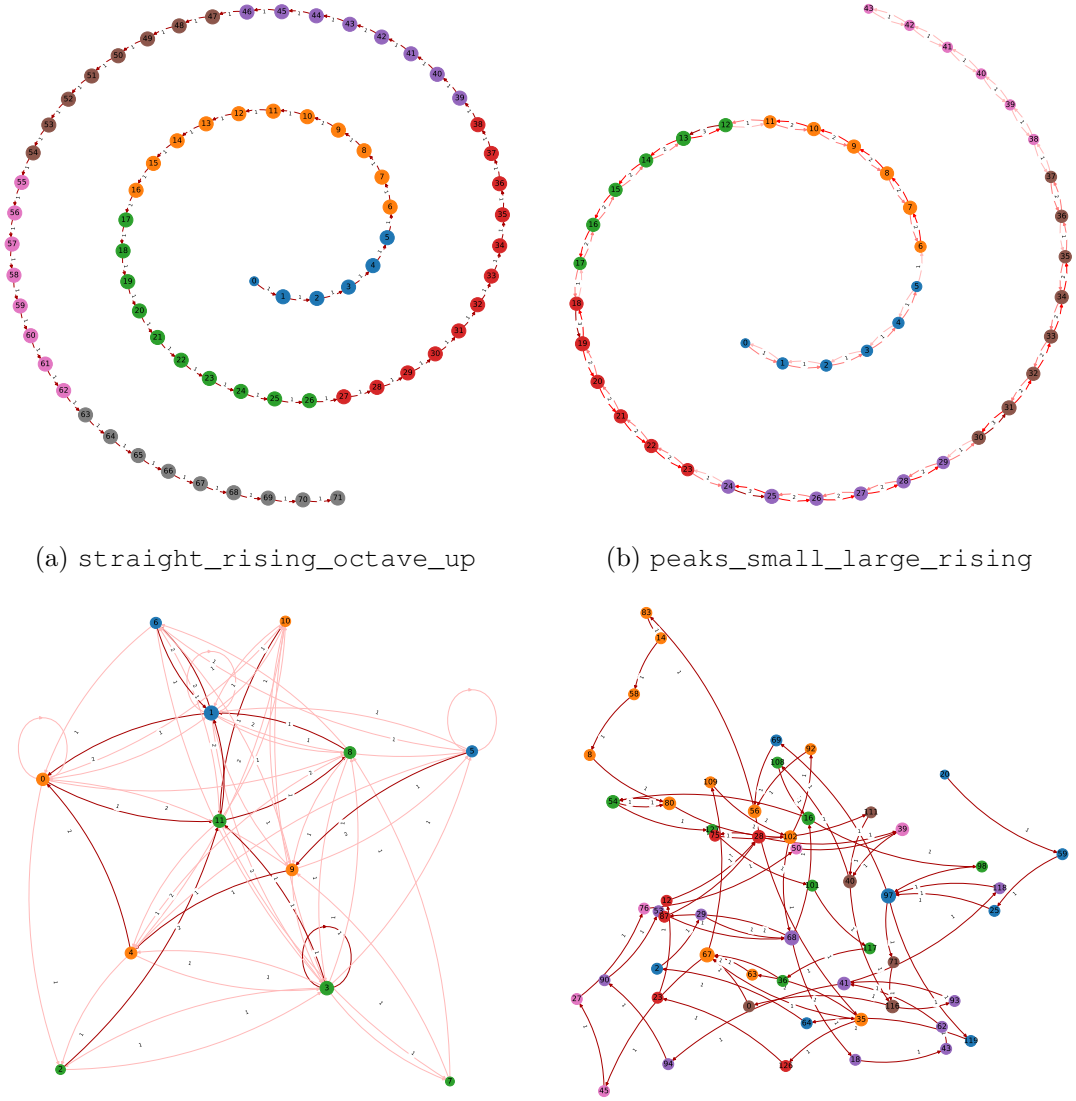
`random_fixed_octave` has the lowest values since there are more edges in proportion to the number of nodes, and each edge has weight 1 or 2. These two facts combined lead to a much lower length of the shortest paths.

Density Density ends up describing the variety of the connections, within a specific number of unique notes. `random_fixed_octave` has a very high density which has an impact on the previously mentioned average weighted betweenness centrality and average weighted closeness centrality, having a very high value in both. While `straight_rising_octave_up` as well

as `random_fully` having many unique notes, but a very low number of edges considering all possible edges between all the existing nodes, have a very low density value.

Modularity `straight_rising_octave_up` and `peaks_small_large_rising` have the highest modularity values because they are both chains. In the case of the first, there is only one outward connection from each node (except the last), with each edge having as weight 1. Each community is just a sequence of consecutive nodes, and they all have more or less the same number of nodes. The latter is a chain with edges in both directions. With the edges on the first few nodes (corresponding to the small peak) have much higher weights than the rest. Which results in having communities on those nodes, and then separate ones on the rest of the nodes.

With the lowest modularity value there is `random_fixed_octave`, which comes from how more diverse and less of a pattern it follows compared to the rest. From its random nature and the number of different connections it has on very few nodes. In contrast with `random_fully` that although also randomly generated ends up as a near chain. We present the four mentioned examples' graphs with the nodes coloured by community in figure 4.3.



(a) `straight_rising_octave_up` (b) `peaks_small_large_rising`

(c) `random_fixed_octave` (d) `random_fully`

Figure 4.3: Graph representations of the synthetic dataset examples with nodes coloured by community

4.3 Music Dataset

Having used our feature set on a synthetic one, we moved on to analysing our original target, which was "real" music.

To test our feature set capabilities we chose a set of songs of three different genres: classic, pop and rock. The goal being to ascertain if it is possible through it to distinguish between songs of the different genres, and if so how. Ideally pinpointing subsets of features that allow us to distinguish between two distinct sets of songs.

As mentioned before, we analysed each song by going through one of its tracks. For this study we picked the track that seemed to better capture the song's voice or melody. That by itself, i.e. choosing the track for each song, is deserving of a more meticulous study. Particularly since not all MIDIs are created the same. Some use one track per instrument, others split the instrument's part into several tracks.

Nevertheless, we believe that even when taking that into account, the results already seem to provide some insights on what it is possible to capture when resorting to complex networks' features.

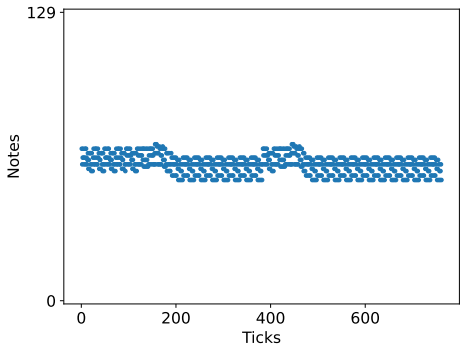
4.3.1 Experimental Analysis

As to better present the differences between the genres, we have studied them in pairs of two.

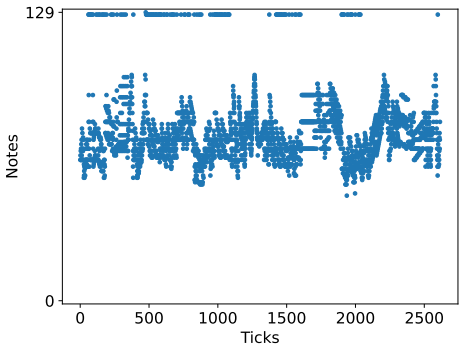
To see how well it fares with real music, after having only tested on a synthetic set which was created to have distinct "songs", we chose to start with a set of classical and pop songs. Since we expected the genres to be the most distinct, and so with the differences more likely to be noticeable in our analysis.

Similarly to what we did with the synthetic dataset, we present the time series representations of each song's chosen track in figure 4.4. Presenting the pop songs representation on the left column, and classical songs on the right one. At a first glance strikes us the different range of notes used on both sets of songs. Pop music being restricted to a noticeably smaller number of notes. We can also see that along the duration of the track there seems to be a greater variation on the classical songs.

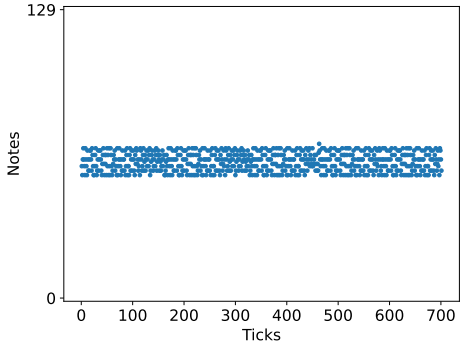
In figure 4.5 we present the graphs of the pop and classical songs. The number of nodes, which represents the number of distinct notes occurring in a track, are much greater on classical songs as previously seen on the time series representation.



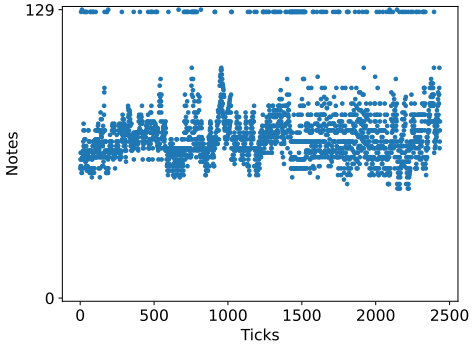
(a) `all_that_she_wants_ace_of_base`



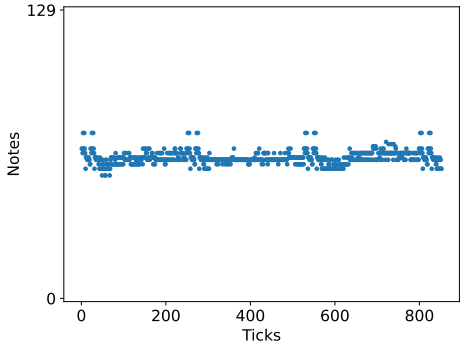
(b) `ballade_in_a_flat_major_op_47_chopin`



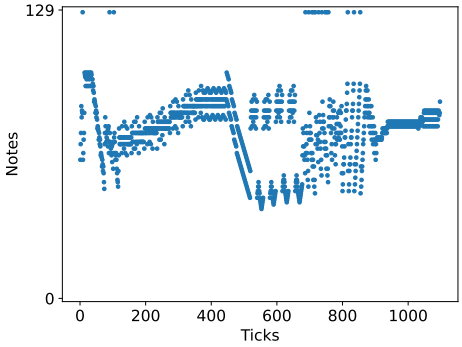
(c) `dont_pull_your_love_grass_roots`



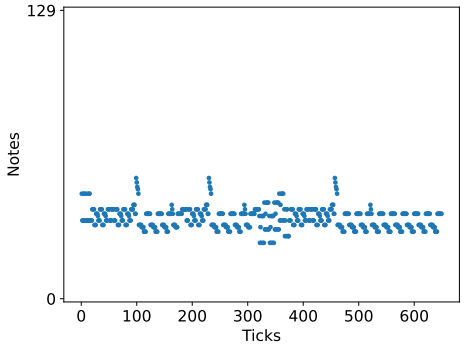
(d) `grande_fantaisie_cubaine_nicolas_espadero`



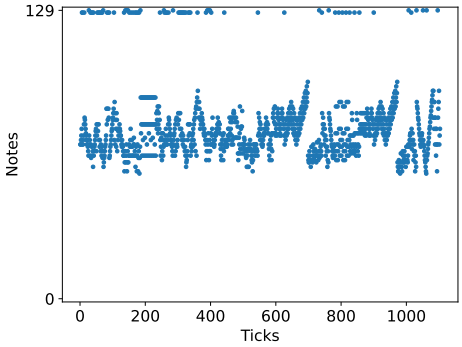
(e) `love_the_way_you_lie_eminem`



(f) `piano_concerto_1st_movement_cadenza_grieg`

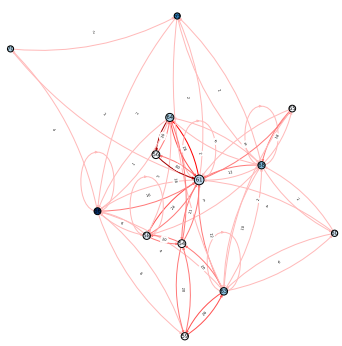


(g) `take_on_me_aha`

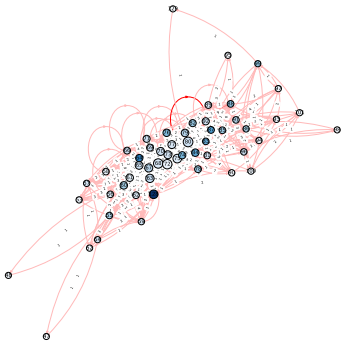


(h) `sonatine_bureaucratique_erik_satie`

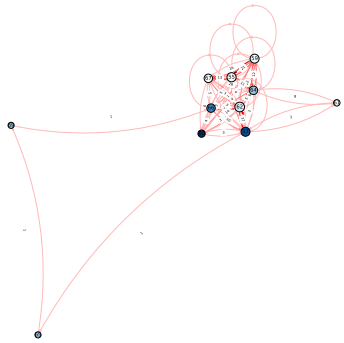
Figure 4.4: Time series representations of the pop and classical songs, respectively, on the left and right columns



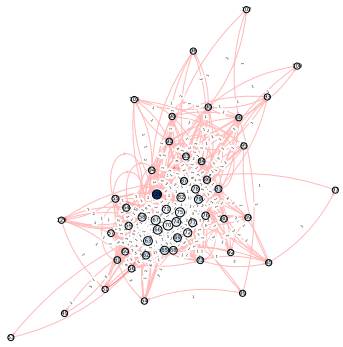
(a) all_that_she_wants_ace_of_base



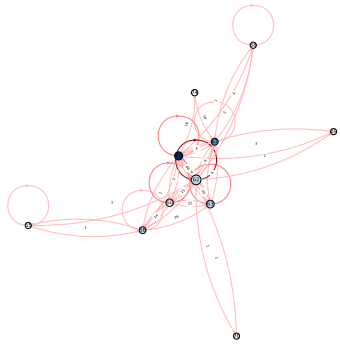
(b) ballade_in_a_flat_major_op_47_chopin



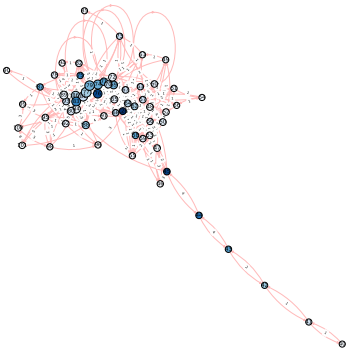
(c) dont_pull_your_love_grass_roots



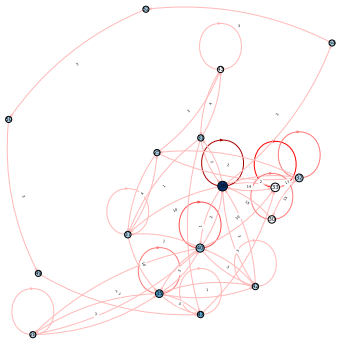
(d) grande_fantaisie_cubaine_nicolas_espadero



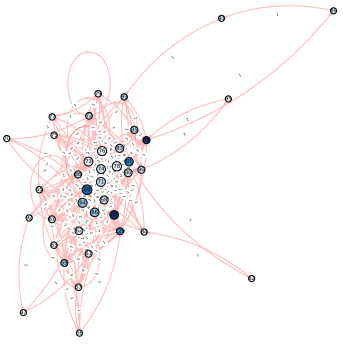
(e) love_the_way_you_lie_eminem



(f) piano_concerto_1st_movement_cadenza_grieg



(g) take_on_me_aha



(h) sonatine_bureaucratique_erik_satie

Figure 4.5: Graph representations of the pop and classical songs, respectively, on the left and right columns

Song	Avg. W. In-Degree	Avg. W. Betweenness Cent.	Avg. W. Closeness Cent.	Avg. Clustering Coef.	Avg. Shortest W. Path Length	Density	Modularity	#Nodes	#Notes
all_that_she_wants_ace_of_base	58.462	16.909	0.094	0.547	12.596	0.378	0.150	13	761
dont_pull_your_love_grass_roots	63.636	14.499	0.171	0.684	6.427	0.573	0.155	11	701
love_the_way_you_lie_eminem	77.364	12.167	0.117	0.800	10.391	0.436	0.414	11	852
take_on_me_aha	38.118	41.637	0.089	0.222	12.316	0.199	0.633	17	649
ballade_in_a_flat_major_op_47_chopin	46.370	88.758	0.356	0.490	2.892	0.233	0.230	54	2481
grande_fantaisie_cubaine_nicolas_espadero	43.204	77.955	0.381	0.504	2.826	0.245	0.233	54	2297
piano_concerto_1st_movement_cadenza_grieg	17.645	148.310	0.247	0.267	4.585	0.109	0.485	62	1081
sonatine_bureaucratique_erik_satie	27.179	58.389	0.369	0.473	2.847	0.234	0.259	39	1028

Table 4.2: Pop and classical songs comparison

Looking now at the feature set we can indeed see some differences. As mentioned before there is a clear difference in the number of unique notes, seen here as number of nodes ($\#Nodes$), used between the two sets of songs with classical songs having a much higher number of unique notes. The pop songs seem to have a higher average weighted in-degree value, despite the classical songs having a higher number of notes ($\#Notes$).

The classical songs have greater average weighted betweenness centrality as well as average weighted closeness centrality. While pop songs seem to have higher average weighted in-degree and average shortest weighted path length.

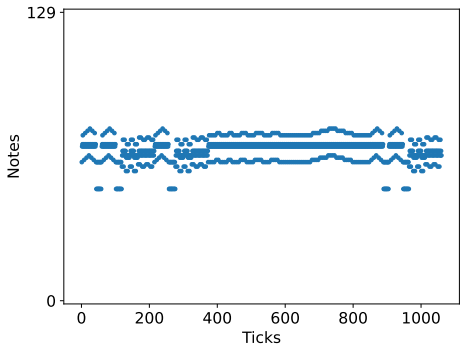
Average clustering coefficient and density display less contrast, but for the most part pop songs seem to have higher values as well.

Modularity values seem to be less similar within the songs of each genre, not being, in this particular case, a reliable feature to distinguish pop and classic. With classical songs having modularity values restricted to a much smaller interval than that of pop songs that seem to be much less represented by this feature.

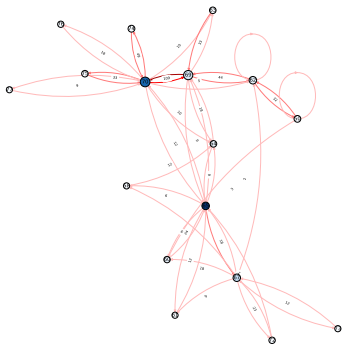
Rock songs To further test our feature set we chose a third genre (rock) which should also be somewhat visibly distinct of the other two. We present the graph and time series representation of each song side-by-side on figure 4.6.

When looking at the time series representation, rock songs seem to display a range of notes in between of that of classical and pop songs. More distinct notes than pop and less than that of the classic genre.

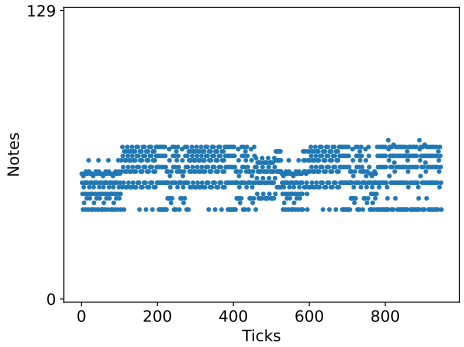
That is also visible in the graphs. But we can also see that rock graphs seem much more spread out than that of pop songs. Which are more closed, and appear to repeat more patterns.



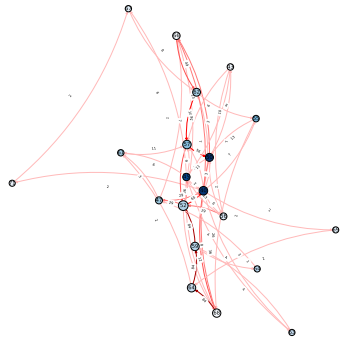
(a) `anyway_genesis` time series representation



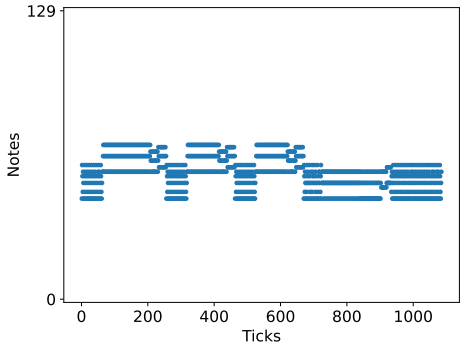
(b) `anyway_genesis` graph



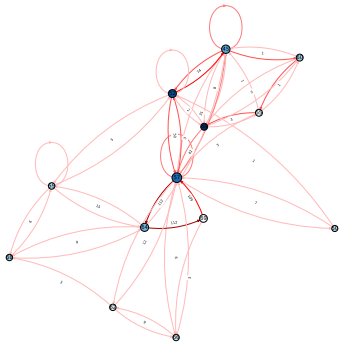
(c) `dirty_deeds_done_dirt_cheap_acdc` time series representation



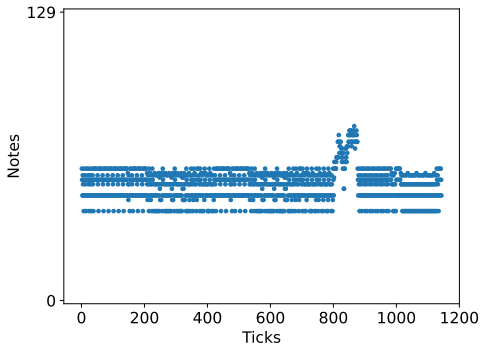
(d) `dirty_deeds_done_dirt_cheap_acdc` graph



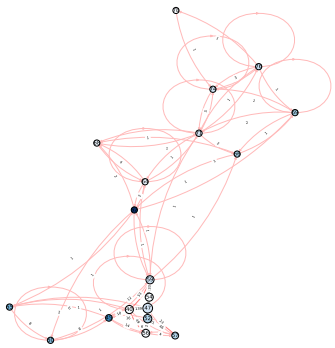
(e) `go_down_acdc` time series representation



(f) `go_down_acdc` graph



(g) `hard_as_a_rock_acdc` time series representation



(h) `hard_as_a_rock_acdc` graph

Figure 4.6: Rock songs: showing time and graph representations side-by-side

In table 4.3 we present the feature values of the chosen songs of the classic and rock genres.

Comparing classical and rock songs we can spot some differences that could also be seen when comparing classical and pop songs. When compared to rock and pop songs, classical songs have lower average weighted in-degree and average shortest weighted path length values, as well as higher average weighted betweenness centrality and average weighted closeness centrality values. The difference in average shortest weighted path length, being however much greater when comparing classical and rock songs.

Not all features present this similarity however, with average clustering coefficient differences being, much more similar in most rock and classical songs, with a stark exception to `dirty_deeds_done_dirt_cheap_acdc` which bears a considerably lower value.

The selected rock songs seem to have a tendency for higher modularity values overall. With `piano_concerto_1st_movement_cadenza_grieg` distinguishing itself from the rest of the songs of the same genre with a much higher modularity value.

With even less contrast than when we compared pop and classical songs, in this case density does not provide any remarkable differences between the two sets. Being much more similar and without any resemblance to a clear tendency of either genre having consistently higher values than the other.

Song	Avg. W. In-Degree	Avg. W. Betweenness Cent.	Avg. W. Closeness Cent.	Avg. Clustering Coef.	Avg. Shortest W. Path Length	Density	Modularity	#Nodes	#Notes
anyway_genesis	62.294	24.706	0.025	0.339	48.085	0.173	0.442	17	1060
dirty_deeds_done_dirt_cheap_acdc	47.350	51.525	0.021	0.153	49.003	0.137	0.327	20	948
go_down_acdc	83.385	22.923	0.080	0.540	21.526	0.308	0.319	13	1085
hard_as_a_rock_acdc	60.105	50.806	0.034	0.405	59.158	0.193	0.231	19	1143
ballade_in_a_flat_major_op_47_chopin	46.370	88.758	0.356	0.490	2.892	0.233	0.230	54	2481
grande_fantaisie_cubaine_nicolas_espadero	43.204	77.955	0.381	0.504	2.826	0.245	0.233	54	2297
piano_concerto_1st_movement_cadenza_grieg	17.645	148.310	0.247	0.267	4.585	0.109	0.485	62	1081
sonatine_bureaucratique_erik_satie	27.179	58.389	0.369	0.473	2.847	0.234	0.259	39	1028

Table 4.3: Rock and classical songs comparison

To conclude our genre-by-genre comparison we analysed the feature values of the genres pop and rock. We then present the table with the comparison in table 4.4. Though not as clearly as the previous two pairs, there are still some noticeable differences between the chosen rock and pop songs.

Pop songs have overall higher average weighted closeness centrality, average clustering coefficient, and density. While rock songs bear higher average weighted betweenness centrality, and average shortest weighted path length. Rock songs also present less disparity in their modularity values when compared to pop songs. With the pop set containing songs with both the highest and lowest modularity value of this set.

The average weighted in-degree value does not seem to be particularly higher in either genre.

Song	Avg. W. In-Degree	Avg. W. Betweenness Cent.	Avg. W. Closeness Cent.	Avg. Clustering Coef.	Avg. Shortest W. Path Length	Density	Modularity	#Nodes	#Notes
anyway_genesis	62.294	24.706	0.025	0.339	48.085	0.173	0.442	17	1060
dirty_deeds_done_dirt_cheap_acdc	47.350	51.525	0.021	0.153	49.003	0.137	0.327	20	948
go_down_acdc	83.385	22.923	0.080	0.540	21.526	0.308	0.319	13	1085
hard_as_a_rock_acdc	60.105	50.806	0.034	0.405	59.158	0.193	0.231	19	1143
all_that_she_wants_ace_of_base	58.462	16.909	0.094	0.547	12.596	0.378	0.150	13	761
dont_pull_your_love_grass_roots	63.636	14.499	0.171	0.684	6.427	0.573	0.155	11	701
love_the_way_you_lie_eminem	77.364	12.167	0.117	0.800	10.391	0.436	0.414	11	852
take_on_me_aha	38.118	41.637	0.089	0.222	12.316	0.199	0.633	17	649

Table 4.4: Rock and pop songs comparison

4.3.2 Clustering Results

A practical, and immediate, application of the proposed feature set, would be to distinguish songs in a more automatic manner. As a simple test we applied a common clustering algorithm to the mentioned music dataset (including all three genres), to evaluate whether it would be a path worth pursuing.

We resorted to the widely used k-means algorithm, which partitions n data entries into k clusters [13]. It does so by iteratively finding clusters that best describe the dataset through a similarity measure. The number of clusters was chosen via calculating the best value of silhouette coefficient, which describes how well divided the data is. I.e., seeing similarity as distance, the silhouette coefficient would measure how close are the entries within each cluster and how far apart are entries of different clusters. We tested for k values ranging from 2 to 6. For our feature values to be reasonably used as similarity measure we normalized each feature (that does not already output values in $[0, 1]$) as mentioned at the beginning of the chapter.

In table 4.5 we present the clustering results of our music dataset. The classical songs were all grouped up in a single cluster isolated from all others. While the pop and rock songs had each one song outside the cluster that contained most of each genre's songs. The cluster 2, which contains three of the four pop songs, contains also one of the rock songs. With a single pop song `take_on_me_aha` remained isolated in cluster 3.

This seems to indicate that the difference between these chosen pop and rock songs is not as great as that of pop and classic, and rock and classic. Which matches our earlier pair-by-pair analysis.

Song	Cluster	Avg. W. In-Degree	Avg. W. Betweenness Cent.	Avg. W. Closeness Cent.	Avg. Clustering Coef.	Avg. Shortest W. Path Length	Density	Modularity
all_that_she_wants_ace_of_base	2	0.621	0.035	0.202	0.609	0.173	0.581	0.000
dont_pull_your_love_grass_roots	2	0.700	0.017	0.418	0.820	0.064	1.000	0.011
love_the_way_you_lie_eminem	2	0.908	0.000	0.267	1.000	0.134	0.706	0.547
take_on_me_aha	3	0.311	0.216	0.190	0.107	0.168	0.194	1.000
ballade_in_a_flat_major_op_47_chopin	0	0.437	0.563	0.931	0.521	0.001	0.267	0.166
grande_fantaisie_cubaine_nicolas_espadero	0	0.389	0.483	1.000	0.542	0.000	0.293	0.173
piano_concerto_1st_movement_cadenza_grieg	0	0.000	1.000	0.630	0.176	0.031	0.000	0.693
sonatine_bureaucratique_erik_satie	0	0.145	0.340	0.969	0.494	0.000	0.270	0.226
anyway_genesis	1	0.679	0.092	0.012	0.287	0.803	0.138	0.606
dirty_deeds_done_dirt_cheap_acdc	1	0.452	0.289	0.000	0.000	0.820	0.061	0.366
go_down_acdc	2	1.000	0.079	0.165	0.598	0.332	0.429	0.350
hard_as_a_rock_acdc	1	0.646	0.284	0.035	0.390	1.000	0.182	0.169

Table 4.5: Music dataset clustering results

Chapter 5

Conclusions

From our tests, we have been able to conclude that in fact features from the song's network topology can be used to capture certain musical aspects, from both synthesized examples and actual songs. As we have used the features to present distinctions between the synthetic examples, as well as between selected songs from different music genres.

Of course, we need to consider that for the music dataset we handpicked tracks for each song, so as to include the tracks that seemed to contain the song's melody (or voice). But that choice influenced, for better or for worse, the results we obtained. Furthermore, this dataset is but a collection of mere 4 song examples of 3 genres, so it is not proof of the effectiveness of our feature set. That said, this does seem to glimpse at its potential.

Our objective is of course to distinguish between real songs in a more autonomous way. Thus finding a better method for choosing the track representing a song would be the first of the follow-up paths that we could pursue to improve our work.

One aspect that we could consider adding to our mapping to improve our work, would be chords. A set of notes being played simultaneously could be seen as a node. Allowing for nodes to contain not just a single note but an entire set of "simultaneously" playing notes. By establishing a threshold we would establish a certain duration where notes that were played within such duration would be considered part of chord. This addition of a new type of node, could provide information that would help us better capture musical aspects, as instruments such as piano or guitar, rely quite heavily on them.

Many more approaches could be followed. We presented a single mapping, and in this we made assumptions that affect our results. Other attributes present in the rich MIDI files can be used to enhance our mapping, or to create other ones. Attributes such as the notes' duration, and velocity.

The quality of the MIDI files used is also relevant. We could benefit from gathering MIDI files that were recorded using similar standards (or perhaps even create our own), so that their quality would not interfere with the representation of each song. Representing all songs in a more

or less consistent way, would ensure that the differences would lay more strongly on the songs themselves and not on added noise or lower quality originated by the creation of the MIDI itself.

MIDIs created by directly capturing notes from instruments will also have their own nuances associated with each person. We could also then, for example, test how similar a person is able to play the same song over and over.

In our study we only used a single track of each song. But we could alternatively resort to creating a multilayer network [14] from the song, where each layer would be mapped to a track. Where notes happening at similar times on different tracks would be connected by interlayer edges. We could analyse specifically piano tracks, which as mentioned, are often composed of two tracks one that mostly represents melody and the other the rhythm. This would give us a way to compare not just the rhythm and melody parts by themselves, but also the music as a whole while still considering them as different types of tracks.

Bibliography

[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[2] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[3] Tanmoy Chakraborty, Sriram Srinivasan, Niloy Ganguly, Sanjukta Bhowmick, and Animesh Mukherjee. Constant communities in complex networks. *Scientific reports*, 3(1):1–9, 2013.

[4] L da F Costa, Francisco A Rodrigues, Gonzalo Travieso, and Paulino Ribeiro Villas Boas. Characterization of complex networks: A survey of measurements. *Advances in physics*, 56(1):167–242, 2007.

[5] Luciano da Fontoura Costa, Osvaldo N Oliveira Jr, Gonzalo Travieso, Francisco Aparecido Rodrigues, Paulino Ribeiro Villas Boas, Lucas Antiqueira, Matheus Palhares Viana, and Luis Enrique Correa Rocha. Analyzing and modeling real-world phenomena with complex networks: a survey of applications. *Advances in Physics*, 60(3):329–412, 2011.

[6] Eric de Silva and Michael PH Stumpf. Complex networks and simple models in biology. *Journal of the Royal Society Interface*, 2(5):419–430, 2005.

[7] Stefano Ferretti. On the modeling of musical solos as complex networks. *Information Sciences*, 375:271–295, January 2017.

[8] Pablo M Gleiser and Leon Danon. Community structure in jazz. *Advances in complex systems*, 6(04):565–573, 2003.

[9] Florian Gomez, Tom Lorimer, and Ruedi Stoop. Complex Networks of Harmonic Structure in Classical Music. In Valeri M. Mladenov and Plamen Ch. Ivanov, editors, *Nonlinear Dynamics of Electronic Systems*, Communications in Computer and Information Science, pages 262–269, Cham, 2014. Springer International Publishing.

[10] Derek Gossi and Mehmet H Gunes. Lyric-based music recommendation. In *Complex networks VII*, pages 301–310. Springer, 2016.

[11] Roger Guimera and Luís A Nunes Amaral. Cartography of complex networks: modules and universal roles. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(02):P02001, 2005.

[12] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[13] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR).

[14] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, September 2014.

[15] Xiao Fan Liu, Chi K. Tse, and Michael Small. Complex network structure of musical compositions: Algorithmic generation of appealing music. *Physica A: Statistical Mechanics and its Applications*, 389(1):126–132, January 2010.

[16] Xiaofan Liu, Chi K. Tse, and Michael Small. Composing Music with Complex Networks. In Jie Zhou, editor, *Complex Sciences*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 2196–2205, Berlin, Heidelberg, 2009. Springer.

[17] Siobhan McAndrew and Martin Everett. Music as collective invention: A social network analysis of composers. *Cultural Sociology*, 9(1):56–80, 2015.

[18] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[19] Federico Pablo-Martí, Ángel Alañón-Pardo, and Angel Sánchez. Complex networks to understand the past: the case of roads in bourbon spain. *Cliometrica*, 15(3):477–534, 2021.

[20] Doheum Park, Arram Bae, Maximilian Schich, and Juyong Park. Topology and evolution of the network of western classical music composers. *EPJ Data Science*, 4:1–15, 2015.

[21] Joan Serrà, Álvaro Corral, Marián Boguñá, Martín Haro, and Josep Ll Arcos. Measuring the Evolution of Contemporary Western Popular Music. *Scientific Reports*, 2(1):521, July 2012.

[22] Lei Wang, Zheng Wang, Chen Yang, Li Zhang, and Qiang Ye. Linux kernels as complex networks: A novel method to study evolution. In *2009 IEEE International Conference on Software Maintenance*, pages 41–50. IEEE, 2009.

[23] Xindi Wang and Ricky Laishram. Understanding Music from Symbolic Representation with Higher Order Networks. page 9, 2020.

[24] Stanley Wasserman, Katherine Faust, et al. Social network analysis: Methods and applications. 1994.

[25] Jian Xu, Thanuka L. Wickramarathne, and Nitesh V. Chawla. Representing higher-order dependencies in networks. *Science Advances*, 2(5):e1600028. Publisher: American Association for the Advancement of Science.