UNIVERSITEIT GENT

FACULTY OF ENGINEERING AND ARCHITECTURE

SONG RETRIEVAL USING DRUM-BASED FINGERPRINTS

Assesment commite:

prof. dr. Tijl De Bie
dr. ir. Cedric De Boom
prof. dr. ir. Dirk Stroobandt

Jaume Martínez Ara
June 2022
Academic Year 2021-2022

Contents

1 Introduction

In the last few decades, it has been discovered the power of data and how important it is to be able to store it for different purposes. Speaking of music, the term of Music Information Retrieval has become extremely important, and despite all the work and progress achieved, it is still a field with lots of things to discover.

As more and more music is created, the necessity of organizing it in an effective way arises. This is when audio fingerprinting appears. Audio Fingerprinting consists in identifying relevant features in an audio track giving it a unique DNA that allows to differentiate that track from others.

One of the most known usages of this technique is for apps such as Shazam, which can detect which song is being played insanely fast. However, there are several applications that are not that straightforward.

If one thinks of DJs, they struggle a lot trying to find songs that can be mixed in a very smooth and cool way. It is true that there are currently some systems that can help them do it, but these systems only focus on easy things such as tempo or the key of the songs.

In this thesis, audio fingerprinting will be used for finding and retrieving similar songs from a database focusing on the drums, so that DJs can find mixable songs based not only on the tempo or the key, but also on the groove. To do so, Dejavu fingerprinting algorithm along with different methodologies will be used.

2 Overview of the application

In order to have a first insight of what this system is about, let's first explain the different softwares it is build on.

First of all, in order for the application to look at the groove of the song, it is necessary to find a way of extracting the drums apart from the rest of the music. The solution for this is Spleeter, a music source separator created by Deezer [1].

Then, as already said at the end of the introduction, Dejavu fingerprinting [9] will do the job for finding a way to uniquely identify every song looking for their most relevant parts. Whenever there is a new input song, the algorithm will look for the songs that match best with the new one, being them the most similar in terms of the groove. Of course, before launching the query, the system will have to spleeter the new song and compute its fingerprints.

The results for a single input are the five most similar songs according to the application, with some information such as the confidence and the time it took for it.

3 Background

The aim of this section is to give a better understanding of the different software tools, techniques and algorithms used along the whole project.

3.1 Audio fingerprinting

It is well known that when speaking of an audio signal one can think about its representation as a spectrogram. [8] For example, the first few seconds of the song “Blurred Lines” by Robin Thickle look like this:

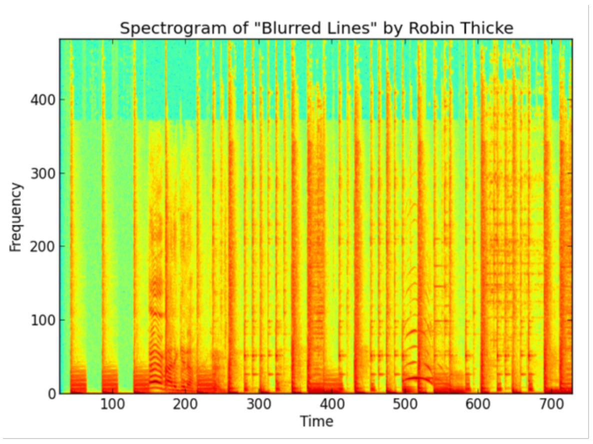


Figure 1: Spectrogram of ”Blurred Lines”

This represents the amplitude as a function of time and frequency. Note that the amplitude is represented continuously by the color (red means higher and green lower) whilst frequency and time are discretized.

The reason of representing the audio as a spectrogram is because it will help give the song a unique way to identify it finding its fingerprints, i.e. DNA.

After having computed those fingerprints, it will be time for song retrieval finding the best matches once the user inputs the song to process.

3.1.1 Fingerprint calculation

In order to determine those fingerprints, the algorithm needs to find the parts where there is more acoustic information: those pairs (frequency, time) that their amplitude is a local maximum. Another reason of finding the greatest amplitudes is because the lower the amplitude the less likely it is for those parts to survive the noise.

There are different ways of finding these local maxima, and the one used in Dejavu fingerprinting algorithm is combining a high pass filter (a threshold in amplitude space) and some image processing technique. A concept of a ”neighboorhood” is needed because a local maximum with only its directly adjacent pixels is a poor peak, as it would not survive the noise.

Thus, the spectrogram shown before with all the peaks marked on it looks like this:

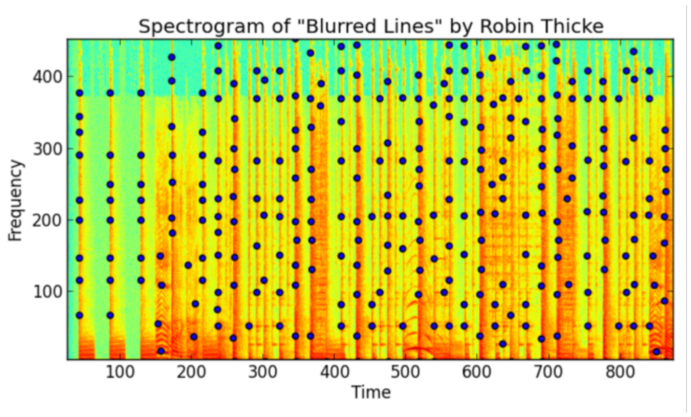


Figure 2: Peaks of ”Blurred Lines”

Next step is to combine these peaks and, using a hash function (the hash used in the Dejavu fingerprinting algorithm is SHA-1), turn them into actual fingerprints. To do so, every peak is combined with its neighbor peaks the following way:

$$fingerprint = hash(frequencies\, of\, peaks, time\, difference\, between\, peaks)$$

Therefore, for every neighbor, a hash is generated with their frequencies (the peak we are dealing with and the neighbor) and their time differences. Thus, if N peaks are found on a certain song and being k the number of neighbors combined, it will have approximately k·N fingerprints in total.

It is important to note that peaks are sorted by time and every peak will be combined with the following k neighbors, being k an arbitrary number. Visually, it can be illustrated like this:

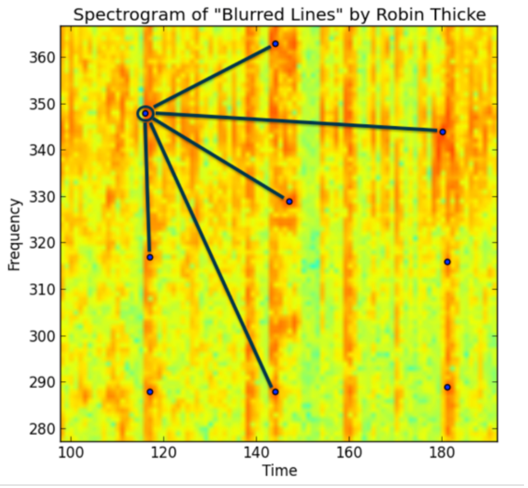


Figure 3: Combination of a peak with its neighbors

The algorithm must be designed taking into account that the more neighbors chosen, the more unique that fingerprints are, but also the less robust to noise.

3.1.2 Song retrieval

Once all the fingerprints are computed, even those from the new input song, the system will compare the fingerprints from the new song to find matches on the database. Once it has all the matches, it is time to what it is called “align matches”.

This part consists in trying to find those matches that preserve time difference. For example, if in the database there is a song which has fingerprint A in second 10 and fingerprint B in second 12; and the new song that is being processed has both fingerprints A and B: fingerprint A in second 3 and fingerprint B in second 5. Both songs have time difference between fingerprint A and B of 2 seconds: those matches are aligned.

However, a problem seems to arise: speed of the song is important. We will go more in depth in later sections and propose a solution to it.

3.2 Energy of a signal

Computing the energy of a signal can help determine the most relevant parts of an audio track. But what do we understand by energy?

By definition, [7] the energy of a signal corresponds to its total magnitude. More precisely, when dealing with audio signals, the energy represents how loud it is. The mathematical definition is the following:

$$E_s = \sum |x(n)|^2$$

3.3 Spleeter

Spleeter is the Deezer source separation library which aims to split an audio track into the different instruments that form it: vocals, drums, bass and the rest. It uses pretrained models written in Python and uses Tensorflow.

The pre-trained models are U-nets. [5] The U-net is a encoder/decoder Convolutional Neural Network (CNN) architecture with skip connections. Spleeter’s models have 12-layer U-nets (6 layers for the encoder and 6 for the decoder). A U-net is used for estimating a soft mask for each source (stem). Training loss is a L1-norm between masked input mix spectrograms and source target spectrograms.

Those models work so well that even professional softwares such as VirtualDJ or Acon Digital use them.

Spleeter can separate an audio file in several ways:

- Vocals (singing voice) / accompaniment separation (2 stems)
- Vocals / drums / bass / other separation (4 stems)
- Vocals / drums / bass / piano / other separation (5 stems)

In this application it is used the second separation as it returns four different audio files with vocals, drums, bass and others.

The models from Spleeter have very high performances and is also very fast as it can perform separation of audio files to 4 stems 100x faster than real-time when run on a GPU.

When using Spleeter it works the following way: an audio file is given as an input as well as the model of separation desired and the path where it will store the results. Then, once it has the separated audio files, the user gets a new folder with the name of the song in the path indicated. Inside, there are as many files as the model chosen returns with the name of the different separations.

Spleeter also allows the user to train the models with their own data, as long as the dataset contains separated files suitable for the training process.

4 Methodology

In this section it will be described how the whole system works giving a good understanding of its different parts.

The application is divided into three main parts: the spleetering, fingerprinting calculation and storing of the whole data (either stretched or not, this will be explained later); the spleetering and fingerprinting calculation of the new input songs (again stretched or not); candidate selection and results.

Here there is a diagram that represents the process in a very high-level way:

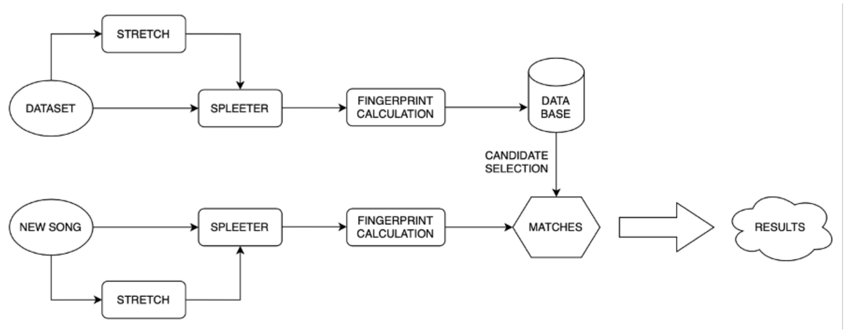


Figure 4: Diagram of how the application works

4.1 Spleetering, fingerprinting and storing the data

As always, data is needed for this whole system to work. The first thing that must be done before jumping into results is preparing a good database with all the songs available. This is very important because the system will only be able to retrieve those songs that have been input as data and their fingerprints have been stored in the database.

However, before fingerprinting all the desired songs, there is one step needed: spleetering. As explained before, the application works based on the groove of the song. Therefore, it is needed to extract the drums apart from the rest of the instruments in order to focus entirely on the groove.

Once all the music is spleetered, there are two options: whether to stretch all the songs to 100 bpm or not. Why is that? If you ever try to shazam a song that is 1.25 times faster than the original one, you will realize that it struggles to find which song it is or it even returns the wrong one. The reason is very simple.

As explained in section 3.1, a fingerprint consists in a pair (frequencies, time difference); therefore, when speeding up or slowing down a song it is easy to realize that the time difference between frequencies will change. Thus, as time difference changes, there are no matches, or they are wrong.

One might think that could not be important, but DJs often play songs that are in a different tempo than the original, as they have to match it with the bpm of the previous and following song.

So, how can the application cope with speeded up or slowed down songs? Indeed, with time-stretching.

If all songs are stretched to, let's say, 100 bpm, the problem should disappear, as now all of them are played at the same speed and regardless of the speed of the new input song the system will be able to retrieve the best songs without struggling because of the speeded up or slowed down songs.

There is, nevertheless, something to consider when time-stretching an audio: the frequency must stay unchanged. There are some libraries such as pyrubberband [4] or librosa that allow to do it.

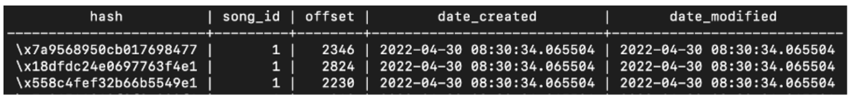
The function of time-stretching of those libraries stretch audio by a rate, so this parameter has to be computed before with a very simple strategy: estimating the tempo with other functions and then rate will be 100/tempo.

Of course, the other option would be not to stretch the audio files and rely on the Dejavu fingerprinting algorithm itself, which turns out to be very resilient to tempo changes according to some experiments that will later be shown and as specified in the paper "Performance of the Dejavu audio fingerprinting framework in music identification in movies" in section 5.2.1. [6]

Next step is fingerprinting the data and store those fingerprints in the database. In this application it is used the PostgreSQL database. [3]

The database schema will contain to tables: fingerprints and songs.

The fingerprints table looks like this:

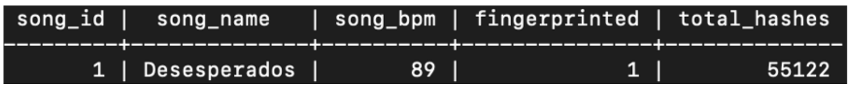


```
         hash           | song_id | offset |        date_created        |       date_modified
------------------------+---------+--------+----------------------------+----------------------------
 \x7a9568950cb017698477 |       1 |   2346 | 2022-04-30 08:30:34.065504 | 2022-04-30 08:30:34.065504
 \x18dfdc24e0697763f4e1 |       1 |   2824 | 2022-04-30 08:30:34.065504 | 2022-04-30 08:30:34.065504
 \x558c4fef32b66b5549e1 |       1 |   2230 | 2022-04-30 08:30:34.065504 | 2022-04-30 08:30:34.065504
```

Figure 5: Table fingerprints

It is important to note that apart from the hash and song ID field, there is a field named offset. It represents the time window from the spectrogram from where the fingerprint belongs. As explained in section 3.1, not all matches are useful, only those that are "aligned". This offset will help know whether they are aligned or not.

The songs table looks like this (in a simplified way):



```
song_id |   song_name    | song_bpm | fingerprinted | total_hashes
---------+--------------+----------+---------------+--------------
       1 | Desesperados |       89 |             1 |        55122
```

Figure 6: Table songs

The fingerprinted field is used internally by Dejavu so that it knows whether a song has been fingerprinted or not. It is set to 0 initially and when it finishes fingerprinted it is set to 1.

4.2 Spleetering and fingerprinting the new input song or clips selected

Once all the dataset is spleetered and fingerprinted, the system is ready to work. Suppose we want to process a song; there are two possibilities: either the song is already stored in the database because it belonged to the dataset, or it is a new song that has never been processed before.

In both cases, the application will spleeter the song to extract the drums and it will stretch the audio to 100 bpm if the dataset was chosen to be stretched. Then, following the same process as for the dataset, next step is to fingerprint the audio and look for the best songs to retrieve.

However, apart from this straightforward method, there exists one other possibility. As one might guess, not all the parts of the song are equally important or relevant. This is when energy can play an important role when it comes to performance.

4.2.1 Energy clip selection

As explained in subsection 3.2, computing the energy of a signal can help determine which are the most relevant parts of an audio. More precisely, when dealing with drums, the most irrelevant parts are those with no drums indeed. To avoid the fact that the algorithm focuses on those parts with no importance whatsoever, a threshold is set to detect silence and eight clips are selected in which there is no "silence".

How do we compute the energy? Librosa [2] has a function called RMS which correspond to the root mean square. Its equation is the following:

$$RMS = \sqrt{1/N \sum |x(n)|^2}$$

Afterwards, when all clips are selected, it is time for fingerprinting each one of them and obtaining the results. For combining these results, a score is assigned to every result. For example, if for one clip the results are A,B,C,D,E in that order, A obtains 5 points, B obtains 4 points, etc. When all eight clips are finished, every song will have a score and the best five are the ones that "win".

4.3 Candidate selection and results

Finally, when all fingerprints are computed, it is time for the results. Nevertheless, are all songs a good match for every other song? The answer is probably not. That is why there needs to be a candidate selection first, before jumping into the results.

The system does not take into account the tempo of the song for filtering the results if there is no indication that says otherwise. However, the main application of this project would be for DJs, as already mentioned, and usually a DJ wants to stick within a range of 6, 10 or at most 16 percent of the tempo when mixing. This is why there exists the possibility to select the candidates that are suitable for that DJ, depending on his or her preferences of mixing.

Thus, the user is asked for a margin of bpm and the system will automatically avoid retrieving songs that are above or below that margin selected. For example, if the song being processed has a bpm of 100 and the user selects a margin of 10 percent, the application will only look for matches in those songs that are within 90 and 110 beats per minute.

Going back to the two situations presented in 4.2, the song can be already in the database or completely new. Let's see what will happen in each of the cases.

In case the song has been stored before, and if the application is built properly and works the way it is supposed to work, the best match will be the exact same song as the input. This is useful in case we did not know which song that was, so it will work as Shazam.

The second case is slightly different. If the song is indeed new, there will be no exact match as the song is completely new to the application, and only those songs that are most similar will be retrieved.

If one thinks further, there is one condition that, in case it is satisfied, both cases converge into the same: when all the existing songs are already processed and stored in the database. It is the case of other applications that their purpose is to only retrieve the exact song, such as the already mentioned Shazam, whose database is formed by all the existing songs and is kept continuously up to date.

5 Experimental evaluation and results

The aim of this section is to evaluate with several experiments the performance of the application by considering some evaluation measures later explained. First of all, we will describe the experimental setup. Afterwards, the results will be presented along with their interpretation.

5.1 Experimental setup

In the following experiments, the system will be evaluated with two datasets, one objective measure, under several assumptions and with three variables that will be combined in order to test the different setups for the application.

5.1.1 Datasets

Two datasets will be used to test the application: the simpler one with only four music genres: reggaeton, rock, techno and hip hop; and the more complex one with six genres: reggaeton, rock, techno, hip hop, drum & bass and house.

The purpose of chosing two datasets is that there is a simple one with less overlapping in tempo, and the complex one with more overlapping so that the system struggles more to return the best matches.

Here there is a picture that illustrates the large dataset:

SONG	ARTIST	TEMPO (bpm)	GENRE
The Phoenix	Alex Stein	130	Techno
My House	Eli Brown	130	Techno
Phobos	Space 92	130	Techno
Dancing	James Hype	125	Techno
Cryptic Speech	UMEK	131	Techno
Por la boca vive el pez	Fito y Fitipaldis	151	Rock
Livin' on a prayer	Bon Jovi	123	Rock
Losing my religion	R.E.M.	125	Rock
Born in the USA	Bruce Springsteen	123	Rock
Highway to Hell	AC DC	116	Rock
Desesperados	Rauw Alejandro	89	Reggaeton
No Me Conoce	Jhay Cortez, Bad Bunny	92	Reggaeton
Lo que pasó pasó	Daddy Yankee	97	Reggaeton
AM Remix	Nio Garcia, J Balvin	86	Reggaeton
El Efecto	Rauw Alejandro	84	Reggaeton
The Next Movement	The Roots	95	Hip Hop
What's the difference	Dr. Dre, Eminem, Xzibit	92	Hip Hop
In Da Club	50 Cent	90	Hip Hop
The Next Episode	Dr. Dre ft. Snoop Dogg	96	Hip Hop
Can I Kick it	A Tribe Called Quest	97	Hip Hop
Summer 91	Noizu	125	House
Feel the vibe	Martin Ikin, Astrotrax	127	House
Sundown	Biscits	125	House
Fur	Endor	127	House
Soul Sacrifice	Dombresky	126	House
Freedom	Bru-C	175	Drum & Bass
Paradise	Bru-C	174	Drum & Bass
CTRL	Grafix	174	Drum & Bass
Take it away	Koven	174	Drum & Bass
Urgency	gyrofield	180	Drum & Bass

Table 1: Dataset

The last two genders (Techno and Drum & Bass) belong only to the large dataset whilst the rest belong to both small and large. Note that Techno, Rock and House overlap or are really close in tempo; Reggaeton and Hip Hop are also very similar in terms of bpm. However, Drum & Bass is alone in its tempo range.

5.1.2 Evaluation measure

In order to evaluate the system, it is needed an evaluation measure that is objective and easy to interpret. There were some possibilities such as asking DJs about the mixes that the application suggests whether they think they could mix those songs in a good way or not. However, that would have taken some time and, besides, it would be very biased to the few DJs that would have been asked and there could be multiple answers. Then, the clearest and most simple evaluation measured seemed to be also very objective: the genre.

Although it can sometimes lead to misleading results, as two songs from two different genres can perfectly be mixed, its simplicity and objectivity make it the chosen measure.

5.1.3 Assumption

In order to make the experiments more consistent, they are made under one important assumption.

The system assumes the tempo estimation is done correctly and therefore the time-stretching algorithm works properly and sets all songs to 100 bpm. In reality, that is not true, as the algorithms for estimating beats per minute might not be perfect and, to make matters worse, the same song can always be interpreted as multiples of the real tempo. For example, if a song is played at 140 bpm, the system can detect tempo as 70 bpm, as it is perfectly suitable for the song.

Therefore, in the datasets used for this thesis, the bpms have been corrected manually in order to ensure the best evaluation possible.

5.1.4 Combination of the techniques

The different techniques we will play with are whether stretching or not the audios, whether using or not the energy clip selection and whether to have the bpm margin for candidate selection or not.

For each of the experiments, the accuracy (based on the evaluation measure explained above) will be computed.

Note that, in order to be able to evaluate all the possibilities, we have created four databases. Two of them fingerprinting the small dataset stretched and non stretched and the other two fingerprinting the large dataset stretched and non stretched.

5.2 Experiments and interpretation of the results

5.2.1 Accuracy of the system

In the first place, we will evaluate the accuracy of the application with the different settings changing whether the audios are stretched or not, whether it uses energy clip selection or not, and whether it uses bpm margin candidate selection or not.

Note that whenever the energy and the bpm are referred in this section, the meaning is ”energy clip selection” for energy and ”bpm margin candidate selection” for bpm. Sometimes it will be shortened.

To proceed to the evaluation, we will focus first on whether the fact of stretching the audio files results in a better performance or not based on the evaluation measure.

Then, by changing the other variables such as the bpm margin for candidate selection or the energy clip selection we will be able to extract some conclusions about their usefulness or not.

The following table is without energy and without bpm selection. The number represents the number of retrieved songs of the same genre as the input over the maximum possible. As there are five songs per genre and the system returns five songs per query, there is a maximum of 25, which would be a perfect accuracy.

Genre	Small Dataset		Large Dataset	
	Non-stretched	Stretched	Non-stretched	Stretched
Techno	10/25	23/25	10/25	18/25
Rock	9/25	11/25	10/25	10/25
Reggaeton	11/25	14/25	10/25	8/25
Hip Hop	12/25	10/25	11/25	11/25
House	-	-	10/25	24/25
Drum & Bass	-	-	16/25	11/25

Table 2: Stretching vs Non-Stretching: NO Energy & NO bpm selection

It is important to note that, in general, for all the genres except for reggaeton in the large dataset and hip hop in the small dataset, the stretching helps get a better accuracy.

However, it just seems to work very nice for techno and house (which might not be a coincidence, as both genres are quite similar). In fact, after going deeper into the results to really see what was happening, turned out that the errors of these two genres were, most of the times, between them: the techno songs sometimes gave as a result some house songs and the other way around.

Actually, if one takes another look at the table, when comparing between small and large dataset, note that techno has a considerable decrease in accuracy, which is due to the addition of house, that the system gets confused.

The following table is without energy clip selection but now with bpm selection with a margin of 16% to the original tempo of the input song:

Genre	Small Dataset		Large Dataset	
	Non-stretched	Stretched	Non-stretched	Stretched
Techno	14/25	24/25	11/25	18/25
Rock	16/25	9/25	14/25	7/25
Reggaeton	15/25	17/25	15/25	17/25
Hip Hop	15/25	19/25	15/25	20/25
House	-	-	13/25	25/25
Drum & Bass	-	-	23/25	25/25

Table 3: Stretching vs Non-Stretching: NO Energy & WITH bpm selection

Again, we can see that the majority of the genres have a better accuracy with the stretched dataset. The same thing as mentioned in the previous table happens now, techno and house work pretty well.

Nevertheless, there are a few other things to stand out: Hip Hop, Drum & Bass and even Reggaeton perform pretty well compared to Table 2 where BPM selection was off. It is actually not a surprise, as with the BPM margin, genres such as Drum & Bass that is alone in its BPM range have suddenly no other possible matches apart from the same genre.

Regarding Hip Hop and Reggaeton, at first it did not make sense. However, when taking a look at the results and noticing that these two genres always got confused with techno or house, and by looking at Table 1 and noticing that they are pretty far away from techno and house in BPM range, it started to make sense.

Next, the table will be representing the values with energy clip selection and without bpm selection:

Genre	Small Dataset		Large Dataset	
	Non-stretched	Stretched	Non-stretched	Stretched
Techno	3/25	17/25	1/25	13/25
Rock	3/25	8/25	7/25	7/25
Reggaeton	10/25	7/25	6/25	6/25
Hip Hop	10/25	9/25	7/25	10/25
House	-	-	3/25	11/25
Drum & Bass	-	-	4/25	5/25

Table 4: Stretching vs Non-Stretching: WITH Energy & NO bpm selection

Once again, we confirm that stretching the audios improve noticeably the performance of the system.

The actual unexpected thing here is the incredible downgrade of the performance of the application when introducing Energy clip selection.

Our initial hypothesis was clear, as explained in section subsubsection 4.2.1. The usage of a technique such as the energy clip selection could improve the results due to the avoidance of silence, as it is an irrelevant part. However, it seems to underperform in a very noticeable way. When stretching the problem diminishes but still we cannot confirm the hypothesis that the energy clip selection technique improves the system. Actually, just by looking at the tables, one can even realize that sometimes the application is not even able to retrieve the exact same songs because the score is less than 5/25, which is terrible.

This might lead to the conclusion that with this technique the system does not have enough information of the new input song to compare with the rest of the database.

The following table is with energy clip selection and with bpm margin selection:

Genre	Small Dataset		Large Dataset	
	Non-stretched	Stretched	Non-stretched	Stretched
Techno	9/25	19/25	5/25	16/25
Rock	14/25	9/25	9/25	7/25
Reggaeton	13/25	15/25	10/25	11/25
Hip Hop	13/25	19/25	14/25	21/25
House	-	-	10/25	15/25
Drum & Bass	-	-	21/25	21/25

Table 5: Stretching vs Non-Stretching: WITH Energy & WITH bpm selection

The table above is useful to observe that, even though the energy seems to underperform the system a lot, when combined with the stretching and the BPM margin selection, the application still performs good enough.

There are, of course, lots of other combinations of the techniques but, provided that with these four table it is enough to extract some conclusions, the rest of the tables will be left in the section 8 in case anyone finds it interesting to have a look.

As the best combination seems to be stretching combined with bpm margin selection, here there are a couple of examples of the output of the system where we can see some things we have explained above:

For the song: Take it Away, which is a Drum & Bass song, it has returned, apart from the exact same song:

- Urgency (Drum & Bass)
- Paradise (Drum & Bass)
- Freedom (Drum & Bass)
- CTRL (Drum & Bass)

The time for the query, assuming the stretching is already done, has been of approximately 11 seconds. It has been a perfect query.

For the song: My House, which is a Techno song, it has returned, apart from the exact same song:

- Phobos (Techno)
- The Phoenix (Techno)
- Feel the Vibe (House)
- Fur (House)

The time for the query, assuming the stretching is already done, has been of approximately 20 seconds. It has not been a perfect query according to the evaluation measure. However, Techno and House are very similar, so they could very easily be mixed.

6 Conclusions and future work

As a final section, we will talk about some general conclusions for the overall work and finally we will give some further considerations and ideas for future work.

In this thesis we have implemented a new system that uses audio fingerprinting for DJ mixing purposes. With the implementation of different techniques and by evaluating their results, we can come to the conclusion that the hypothesis that time-stretching songs to a common tempo helps improve the performance and makes the system more robust against tempo issues is confirmed due to the different experiments carried out.

However, we have not confirmed the hypothesis that energy clip selection is a way of focusing more on relevant parts of the audio, as the performance of the system not only did not improve, but it decreased.

As for the future work, it would be great to improve the accuracy of the algorithm by maybe modifying how fingerprints are calculated making the algorithm more drum suitable.

Besides, having an actual application with a user interface would be interesting for commercial purposes.

Last but not least, taking the system to a much higher level testing it with large databases would make the evaluation much more reliable.

7 Bibliography

References

[1] Deezer. Spleeter github. URL `https://archives.ismir.net/ismir2019/latebreaking/000036.pdf`.

[2] Librosa. URL `https://librosa.org/doc/0.9.1/generated/librosa.effects.time_stretch.html#librosa.effects.time_stretch`.

[3] PostgreSQL. URL `https://www.postgresql.org`.

[4] B. M. Pyrubberband. URL `https://pyrubberband.readthedocs.io/en/stable/generated/pyrubberband.pyrb.time_stretch.html`.

[5] D. R&D. Spleeter: a fast and state-of-the art music source separation tool with pre-trained models. URL `https://archives.ismir.net/ismir2019/latebreaking/000036.pdf`.

[6] N. Struharová. Performance of the dejavu audio fingerprinting framework in music identification in movies. URL `https://repository.tudelft.nl/islandora/object/uuid:1ba40730-ad68-40b7-a593-c8dbfad3cb1a/datastream/OBJ/download`.

[7] Wikipedia. URL `https://en.wikipedia.org/wiki/Energy_(signal_processing)`.

[8] WillDrevo. Audio fingerprinting with python and numpy. URL `https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/`.

[9] Worldveil. Dejavu github. URL `https://github.com/worldveil/dejavu`.

8 Appendix

Genre	Small Dataset		Large Dataset	
	Without energy	With energy	Without energy	With energy
Techno	24/25	19/25	18/25	16/25
Rock	9/25	9/25	7/25	7/25
Reggaeton	17/25	15/25	17/25	11/25
Hip Hop	19/25	19/25	20/25	21/25
House	-	-	25/25	15/25
Drum & Bass	-	-	25/25	21/25

Table 6: Energy vs NO Energy clip selection: WITH Stretching & WITH bpm selection

Genre	Small Dataset		Large Dataset	
	Without energy	With energy	Without energy	With energy
Techno	14/25	9/25	11/25	5/25
Rock	16/25	14/25	14/25	9/25
Reggaeton	15/25	13/25	15/25	10/25
Hip Hop	15/25	13/25	15/25	14/25
House	-	-	25/25	10/25
Drum & Bass	-	-	23/25	21/25

Table 7: Energy vs NO Energy clip selection: NO Stretching & WITH bpm selection

Genre	Small Dataset		Large Dataset	
	Without energy	With energy	Without energy	With energy
Techno	23/25	17/25	18/25	13/25
Rock	11/25	8/25	10/25	7/25
Reggaeton	14/25	7/25	8/25	6/25
Hip Hop	10/25	9/25	11/25	10/25
House	-	-	24/25	11/25
Drum & Bass	-	-	11/25	5/25

Table 8: Energy vs NO Energy clip selection: WITH Stretching & NO bpm selection

Genre	Small Dataset		Large Dataset	
	Without energy	With energy	Without energy	With energy
Techno	10/25	3/25	10/25	1/25
Rock	9/25	3/25	10/25	7/25
Reggaeton	11/25	10/25	8/25	6/25
Hip Hop	12/25	10/25	11/25	7/25
House	-	-	10/25	3/25
Drum & Bass	-	-	16/25	4/25

Table 9: Energy vs NO Energy clip selection: NO Stretching & NO bpm selection

Genre	Small Dataset		Large Dataset	
	Without bpm	With bpm	Without bpm	With bpm
Techno	23/25	24/25	18/25	18/25
Rock	11/25	9/25	10/25	7/25
Reggaeton	14/25	17/25	8/25	17/25
Hip Hop	10/25	19/25	11/25	20/25
House	-	-	24/25	25/25
Drum & Bass	-	-	11/25	25/25

Table 10: BPM selection vs NO BPM selection: WITH Stretching & NO Energy

Genre	Small Dataset		Large Dataset	
	Without bpm	With bpm	Without bpm	With bpm
Techno	10/25	14/25	10/25	11/25
Rock	9/25	16/25	10/25	14/25
Reggaeton	11/25	15/25	10/25	15/25
Hip Hop	12/25	15/25	11/25	15/25
House	-	-	10/25	13/25
Drum & Bass	-	-	16/25	23/25

Table 11: BPM selection vs NO BPM selection: NO Stretching & NO Energy

Genre	Small Dataset		Large Dataset	
	Without bpm	With bpm	Without bpm	With bpm
Techno	17/25	19/25	13/25	16/25
Rock	8/25	9/25	7/25	7/25
Reggaeton	7/25	15/25	6/25	11/25
Hip Hop	9/25	19/25	10/25	21/25
House	-	-	11/25	15/25
Drum & Bass	-	-	5/25	21/25

Table 12: BPM selection vs NO BPM selection: WITH Stretching & WITH Energy

Genre	Small Dataset		Large Dataset	
	Without bpm	With bpm	Without bpm	With bpm
Techno	3/25	9/25	3/25	5/25
Rock	3/25	14/25	7/25	9/25
Reggaeton	10/25	13/25	6/25	10/25
Hip Hop	10/25	13/25	7/25	14/25
House	-	-	3/25	10/25
Drum & Bass	-	-	4/25	21/25

Table 13: BPM selection vs NO BPM selection: NO Stretching & WITH Energy