EXPERIMENTS WITH THE SHAZAM MUSIC IDENTIFICATION ALGORITHM

BY

FANGJIAN FLORA XIAO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Research Associate Professor Margaret Fleck

ABSTRACT

The motivation of this study is to identify music without the original recording. The existing solutions tackle variations in some properties such as background sound and white noise, but the identification of samples containing large variations in key, tempo, ornamentation, and harmonization remains largely unsolved.

This study takes an existing algorithm and uses an existing data set to explore the parameters required for successful identification, as well as variations in key. The findings show a simple way to identify and normalize the key of a sample. Future work will tackle tempo and ornamentation challenges.

To my parents, my mentors, my friends, and my colleagues.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 MY MOTIVATION

Imagine having a song stuck in your head, but being unable to remember the song name. Maybe the lyrics were elusive or unclear, or maybe it never had any in the first place. Well, I have one, but I wish mine was such a popular song that humming it to my music major friends would have solved my problem.

I ran into one person who knew the name of my mystery song when I was in 7th grade, but 7th grade has long been forgotten. I didn't write down the name of the song, and the first place I'd heard it was a remix in a different language. I've been searching ever since, and I hope one day that I can find it by humming to my phone.

Meanwhile, I've tried all sorts of different music recognition software. Shazam uses an efficient algorithm for exact tracks [1]. Several other apps appear to do the same, but there does not appear to be a published algorithm for them [2, 3, 4].

The Musipedia website offers several different methods to search melodies, including virtual pianos, microphone input, rhythm search, and contour search. The virtual pianos take mouse clicks on a Javascript piano, or midi input from an external controller; the microphone inputs are raw sound; the rhythm search is tapped into the input using the space bar on a computer keyboard; the contour search uses Parson's code, which is requires the user to compare each individual note to the one before it and sometimes to judge whether or not grace notes are entered.

Unfortunately, the virtual pianos, microphone inputs, and rhythm search all had the same problem: you have to have the same timing as the original. The contour search was successful for many very popular songs, but was confusing to use.

This research problem, music recognition, is my chance to take a look at the Shazam algorithm and try it out. Hopefully, I will end up with something I can use to locate my mystery song.

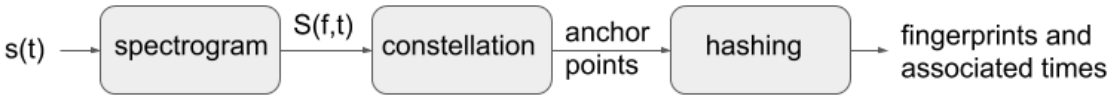


Figure 1.1: The preprocessing that creates fingerprints

1.2 HOW SHAZAM WORKS

Avery Wang's 2003 paper,"An Industrial-Strength Audio Search Algorithm," described the underlying technique Shazam uses to recognize exact tracks [1]. For each piece or song, Shazam takes the spectrogram of the signal, determines the relative peaks, and plots the peaks as a cleaner version of the spectrogram. A spectrogram is a plot of frequency over time, and the designation of "relative peaks" is not specified.

Then, Shazam chooses a large set of anchor points throughout the song, and creates pairs with each anchor point and a set of neighboring points. The neighboring points are all of the points within an area following the anchor point, within some range of time and within some range of notes above and below the anchor point. All of the pairs for a song are then each made into a hash, or fingerprint, and saved in a database along with their absolute time of occurence for later use. The average number of pairs per anchor point is referred to as the fan-out factor. See Figure 1.1 for the preprocessing data flow.

When searching, the same technique is used to generate hashes for the query. Then, for each song that contains matching hashes, the matching hashes are plotted: the query's time on the vertical axis, the song's time on the horizontal axis, and one plotted dot per hash match. If a diagonal of dots appears, that means the query's hashes coincides with the song sequentially along that period of time, then the song is deemed a result. For a non result, only a scatter of matching hashes would occur.

The Shazam paper gives a few hints for anchor point choice, expected fan-out factor, and diagonal finding, but exact implementation details are not included. I will use this opportunity to explore various metrics used to choose anchor points, keeping the effect of tempo and note density in mind.

1.3 WHY IS THIS HARD?

There are many properties that can differ between two tracks of the same song. Any two recordings of the same song by two different artists are almost guaranteed to be different, in timbre and sometimes in instrument. Even the same artist may perform a song at a slightly different key or tempo, with different ornamentation, or even with different harmonization depending on his or her mood. Ornamentation is a set of notes that are added close in pitch and time to the song without changing the structure; an example can be found in Figure 1.2. Harmonization and orchestration are types of accompaniments that can be added to a song's melody, usually to make it sound better.

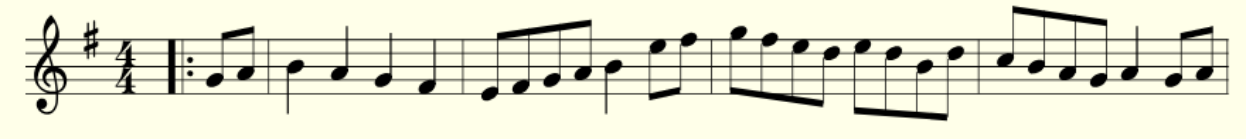
Even holding the above properties constant by using two instances of the same recording in two different environments, the background and incidental noise can cause a mismatch between the tracks. Shazam solves this case. The exact track is identified through peaks and their relationships, and the algorithm is developed specifically to match songs of the same key and tempo, regardless of background (and sometimes foreground) noise. In some instances, editing of the ornamentation may result in matches as well, but the challenge in changing ornamentation is more the reproduction of the exact key and tempo, not the ornamentation itself.

The problem that Shazam doesn't solve is when the same artist changes the key or the tempo. A different harmonization would also register as a different song entirely. It may be up for debate whether or not it is a different song given that many remixes sound completely different than the original. Remixes aside, even perceptible but small changes in key or tempo cannot be matched by Shazam.

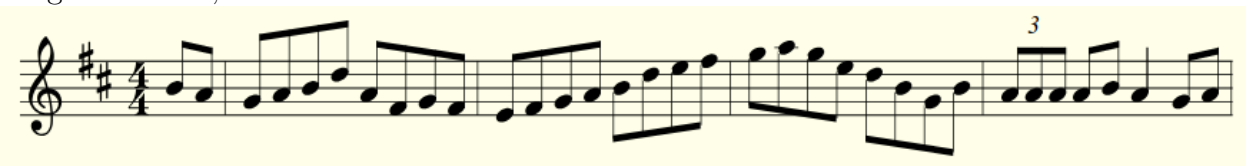
Given two recordings of the same song by two different artists, we may face two different instruments, or at least two different voices. Studies on the signals of human voices expose the difficulty of this subject in music transcription [5]. Voices may sometimes differ so much that each voice can be thought of as a different instrument.

This is then not a signal processing problem, but rather a music theory problem. The signal processing and hashing algorithm presented by Shazam is sufficient for identifying exact tracks, but it cannot match songs that differ widely in timbre, key, tempo, ornamentation, and harmonization.

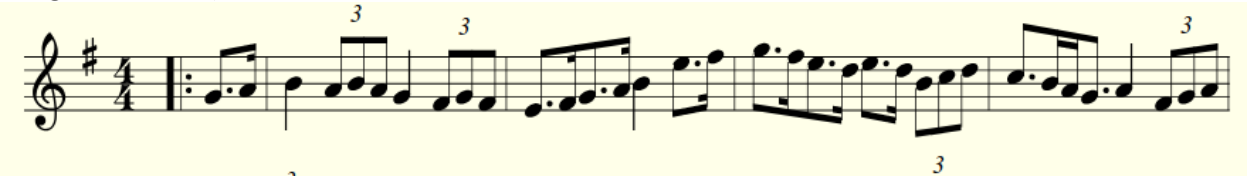
“Rights of Man,” variation 1:



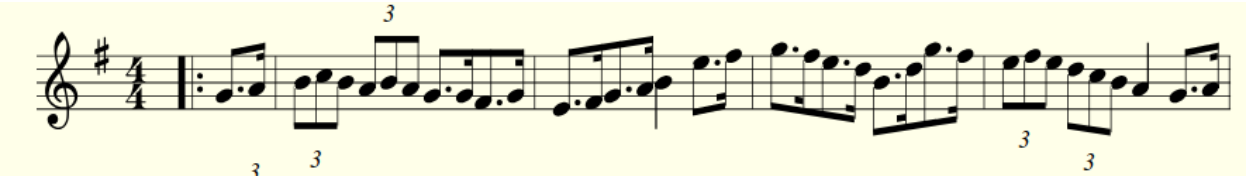
“Rights of Man,” variation 2:



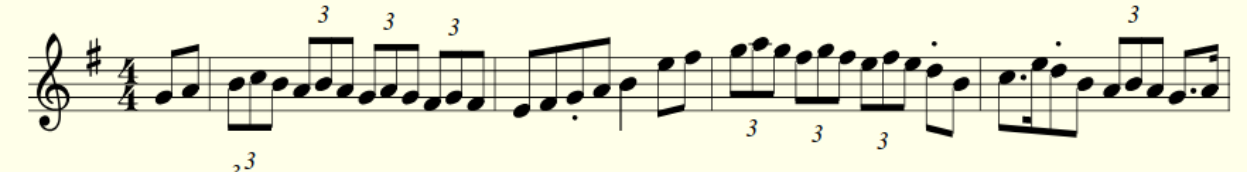
“Rights of Man,” variation 3:



“Rights of Man,” variation 4:



“Rights of Man,” variation 5:



“Rights of Man,” variation 6:

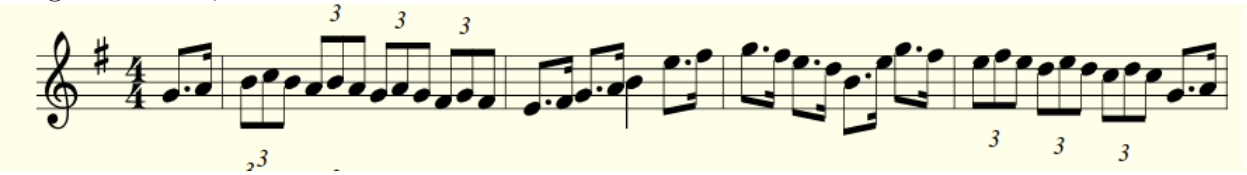


Figure 1.2: Example showing different ornamentation for the same song [6]

1.4 RELATED WORK

Related work can be broken up into three different categories: signal processing, genre categorization, and musical structure matching. Each topic explores an interesting aspect of the music recognition problem. Although there is overlap between the three, they are still distinct topics.

The first of these is signal processing, which I chose to eliminate from my experiments in this thesis. The topic has been tackled from many different angles, and many publications in the conference proceedings of the International Society of Music Information Retrieval (ISMIR) address the topic as "transcription." Techniques include Non-negative Matrix Factorization (NMF) [7], Deep Canonical Correlation Analysis (DCCA) [8], and various neural networks [9, 10]. A cursory Google search will show the trending transcription software in the music industry.

Genre categorization is also a very popular topic, and that is not surprising considering it can be monetized through music recommendation services [11]. Like signal processing, many different techniques have been used to attempt genre categorization, including deep learning [12]. However, this is not the problem I am trying to solve.

Actual musical structure recognition is an ill-defined problem and doesn't seem to be explored much. The only publicly accessible papers for any kind of music recognition were either the Shazam paper [1] or similar fingerprinting algorithms [13, 14]. However, Midomi [3], SoundHound [2], Google Music Search [4], and Musipedia [15] all offer their own music recognition services. I did not find any papers documenting their algorithms.

1.5 MY APPROACH

For simplicity, I decided to use MIDI inputs. MIDI uses numbers to indicate note name, velocity (sometimes equated with loudness), and start/end time for each note. This clear format allowed me to focus on the algorithm without having to worry about signal processing. Signal processing involves converting the raw wave signals into frequencies, then figuring out where the notes are and where they started/ended. It's surprisingly difficult, as I found out while working on my final project from CS 598 PS last fall [16].

Using the note names, I could assume correct pitch, intensity, and duration. I used Python to read the files and run experiments. I originally planned on making my own datasets with a MIDI controller, but due to time and musical skill limits, I quickly realized this was not feasible. I searched online for datasets, selected representative samples in Section 2.1, and designed experiments that could use the representative samples to examine the Shazam algorithm choices in Section 2.2.

I looked for datasets of MIDI files and chose one that contained many duplicates of songs that I could use to compare against. I took each song that had multiple files and generated a heatmap, using colors to show the note duration and velocity. These were helpful in determining what aspects of the song were important in the identification algorithm. Some results were unexpected, as described in Section 2.3.

For this study, I did not consider key changes within the same song. This may involve windowing the song to check snippets instead of the whole song. To window the song is to chop it up into pre-determined lengths and evaluate the snippets individually. This is worth considering at some point; I have included a guide in Section 3.1 with ideas on things I would like to explore in the future but considered out of scope for this thesis.

CHAPTER 2: EXPERIMENT

2.1 DATASETS I USED

I used the dataset "clean_midi" from the Lakh MIDI Dataset [17, 18], which contain labelled MIDI files indicating their artist and title. I searched through it for any numbered duplicate filenames, and created a list of filenames for each.

MIDI files are simple, but they are constructed of messages rather than objects. I made a fork, or copy, of the midi2ly library by gin66 on GitHub [19] to convert the messages into Python objects, to keep track of pieces, tracks within pieces, and notes within tracks. My fork contains the code for that [20].

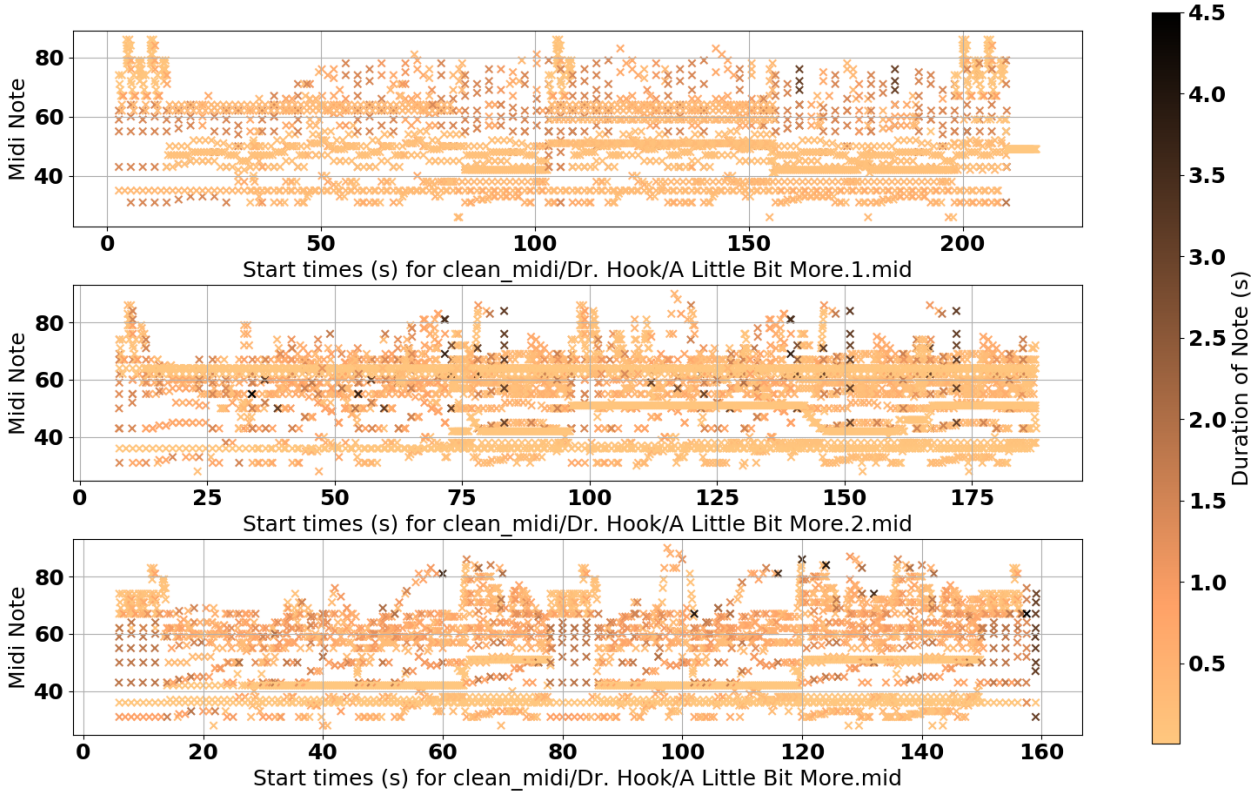


Figure 2.1: Three different files with numbered, duplicate filenames.

2.1.1 Spectrograms

Since I am working with MIDI files, I am not actually plotting real spectrograms. Real spectrograms plot the frequency on the vertical axis and time stamps on the horizontal axis. MIDI uses note numbers instead of frequencies to show pitch, so I put the note numbers on the vertical axis and time stamps on the horizontal axis, which is similar, so I call my plots spectrograms anyways.

There were many sets of files that were named similarly. For each set of similarly named files, I made spectrograms of all the files next to each other to compare the files. Figure 2.1 is a spectrogram comparison for one of these songs. Note that the time units are in seconds.

For each song, I plotted all the files with the same song name along the same time axis. However, some files simply had a different tempo or number of repeats, so the results were different per song. These plots had a more interesting feature when I plotted them using a heatmap for duration or velocity. When the axes were independent, the relative durations seemed to match up, except for some select bass notes. The velocities matched very similarly between files of the same song.

When looking at the spectrograms, I noticed that many contained the same three most popular notes, so in Section 2.2 I decided to take a look at note popularity when choosing anchor points.

In addition to plotting duration on a spectrogram, I also plotted the changes in duration compared the average of notes around it. The plots I tried were comparing against the averages within 1 second, within 2 seconds, within 3 seconds, within 5 seconds, and within 10 seconds of each note. Normalizing plots for changes in duration actually turned out to be not all that helpful, since the plots looked similar to simply plotting the durations themselves. I concluded that there was significant enough difference in duration even within 1s that normalization had minimal effect on the structure of the music.

2.2 ANCHOR POINT CHOICE

I considered three different ways to choose my anchor points:

1. Choose all points, or notes
2. Choose the three most popular named notes by duration
3. Choose the points that were longer than their neighbors
4. Choose the points with higher velocity

The first was mentioned in the Shazam paper. The second, I expected to be the tonic, dominant, and subdominant, or the most musically significant notes, of any piece in a major key. The third, I expected to be beginning or ending notes in a measure. Some songs were very dense, so I concentrated on the last two in an effort to reduce computation resource requirements.

2.2.1 Three Popular Notes

I tried highlighting the top three notes, expecting to see the tonic, dominant, and subdominant of the songs. I expected these notes to be the most frequent, or at least to take up the most duration, due to their responsibility in providing a sense of closure to a phrase. It made sense to expect these notes to happen more frequently or at least last longer, such as at the end of a stanza.

Unfortunately, the top three were inconsistent and most of the time didn't even match up with the expected tonic, dominant, and subdominant notes. I then plotted the histograms of the named notes to see if there was some way to understand the note distribution. At first, I simply plotted the notes in Figure 2.10, found in Section 2.3. I fiddled around with this a little, and realized that I could weight the distribution by duration, which clarified the similarities between files of the same song, like in Figure 2.11, also in Section 2.3.

2.2.2 Longer Than Neighbors

In an attempt to choose points in a more sparse way, I considered choosing points that were longer than their neighbors. My reasoning was that longer notes would have more emphasis, and would be more likely to be similar across samples. I took a plot of relative length to the surrounding 1 seconds, 2 seconds, 3 seconds, 5 seconds, and 10 seconds, and realized that there was no perceptible difference between those plots and the spectrograms themselves, so that put me right back at square one.

2.2.3 Higher Velocity

I also took a look at the velocity of the notes, expecting the higher velocity notes to be more important and thus more replicable. I plotted the spectrograms with the velocity in the heatmap colors in Figure 2.2; the velocity tended to only show on higher notes or during specific sections, but with no obvious way to distinguish the "landmark" notes using velocity.

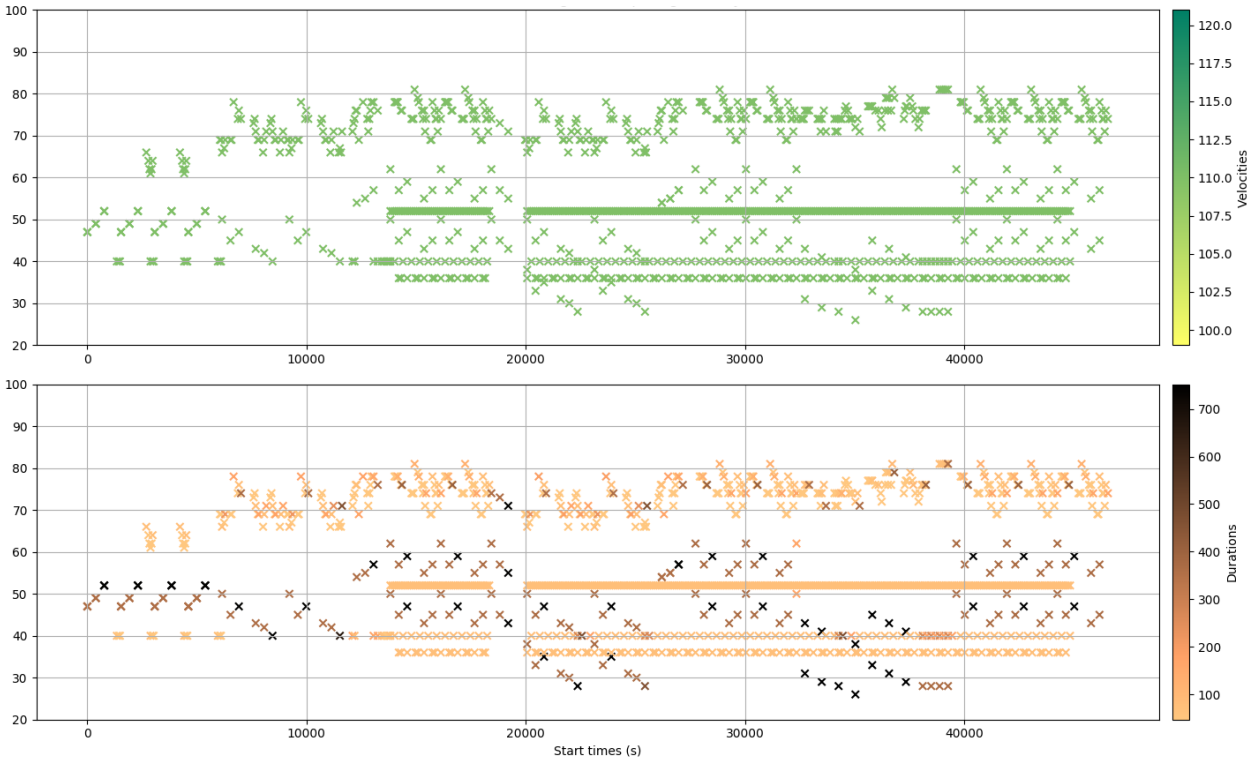


Figure 2.2: Spectrograms of "Don't Want To Miss A Thing" with heatmaps of velocity and duration.

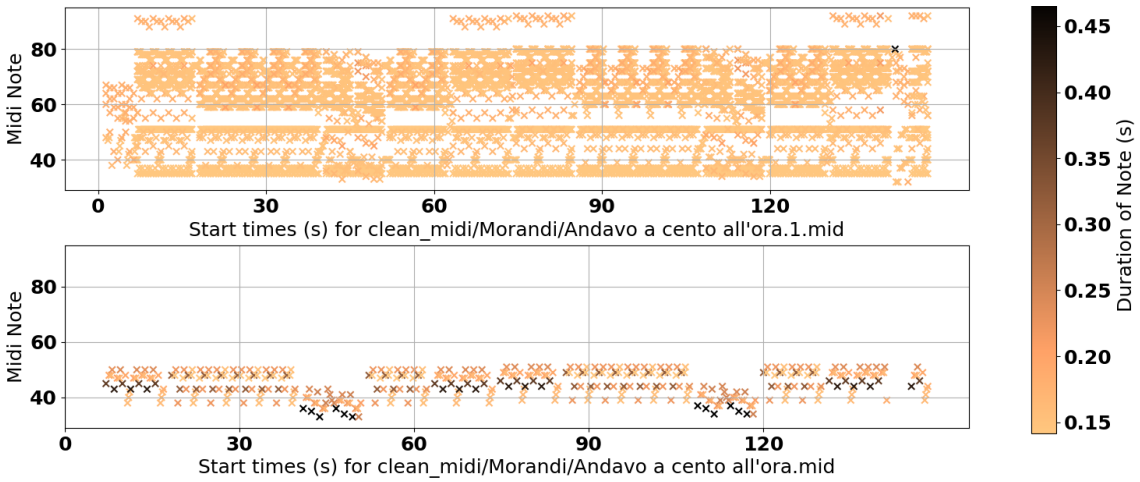


Figure 2.3: Spectrograms of files named "Andavo a cento all'ora"

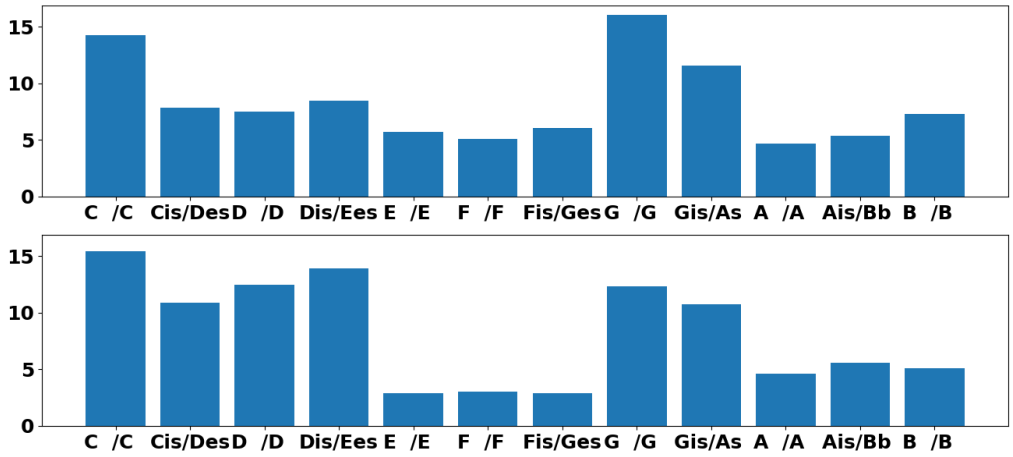


Figure 2.4: Histograms of Notes for "Andavo a cento all'ora" by number of notes.

2.2.4 Using All Notes

Although some of these songs might be reliably recognized through choosing a select number of anchor points, it may be feasible to use all notes as anchor points. This creates a graph that is more dense, but since the second point for each edge chosen is determined by relatively narrow boundaries, the fan-out factor can be kept fairly low, and thus the density of the graph is limited by the note density of the file in question.

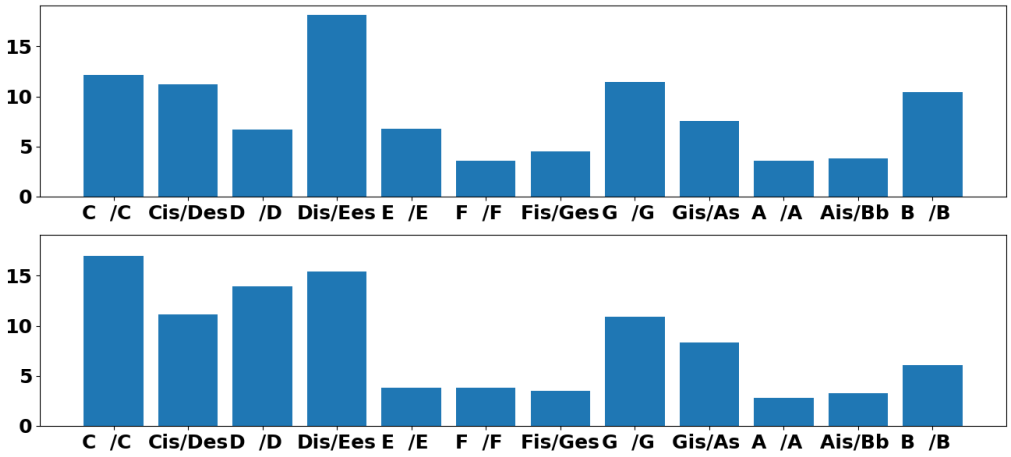


Figure 2.5: Histograms of Notes for "Andavo a cento all'ora" weighted by duration.

2.2.5 Bass and Treble Distinction

After taking a look at the spectrograms I plotted, I noticed that some files would be very sparse due to containing notes that were only above or below Middle C, or note number 60. I made a guess for "bass only" or "treble only" based on the number of notes above or below Middle C. I counted the number of notes above note number 60 and below note number 60; if 80 percent or more of the notes were on either side, I would mark them as "bass only" for below, or "treble only" for above.

Bass and treble tracks have some unique characteristics that distinguish them from tracks that contain both. Bass lines tend to act as a musical anchor and generally tend to be more repetitive, with the most popular notes of the song dominating its time. I would hazard a guess that the bass lines were karaoke tracks. Treble tracks tend to carry the melody of the song, giving it a musical contour, but not necessarily providing much anchor for note popularity.

Figure 2.3 shows two versions of a song, one containing the full song range and one containing mostly or only bass notes. Despite the range difference, their named note histograms still look very similar, in both Figure 2.4 and 2.5. This shows how powerful the histogram technique can be.

Out of the 711 songs that had more than one file, I was able to parse the MIDI files for 646 songs. There were 1783 files that existed for those 646 songs. Of these, 247 files across 180 songs contained bass only; 62 files across 49 songs contained treble only; 1 file contained only Middle C occurring in various rhythms across the whole file. This is interesting because the bass lines are very likely to contain anchoring notes for the song, and the treble only files are very likely to contain the melody of the song.

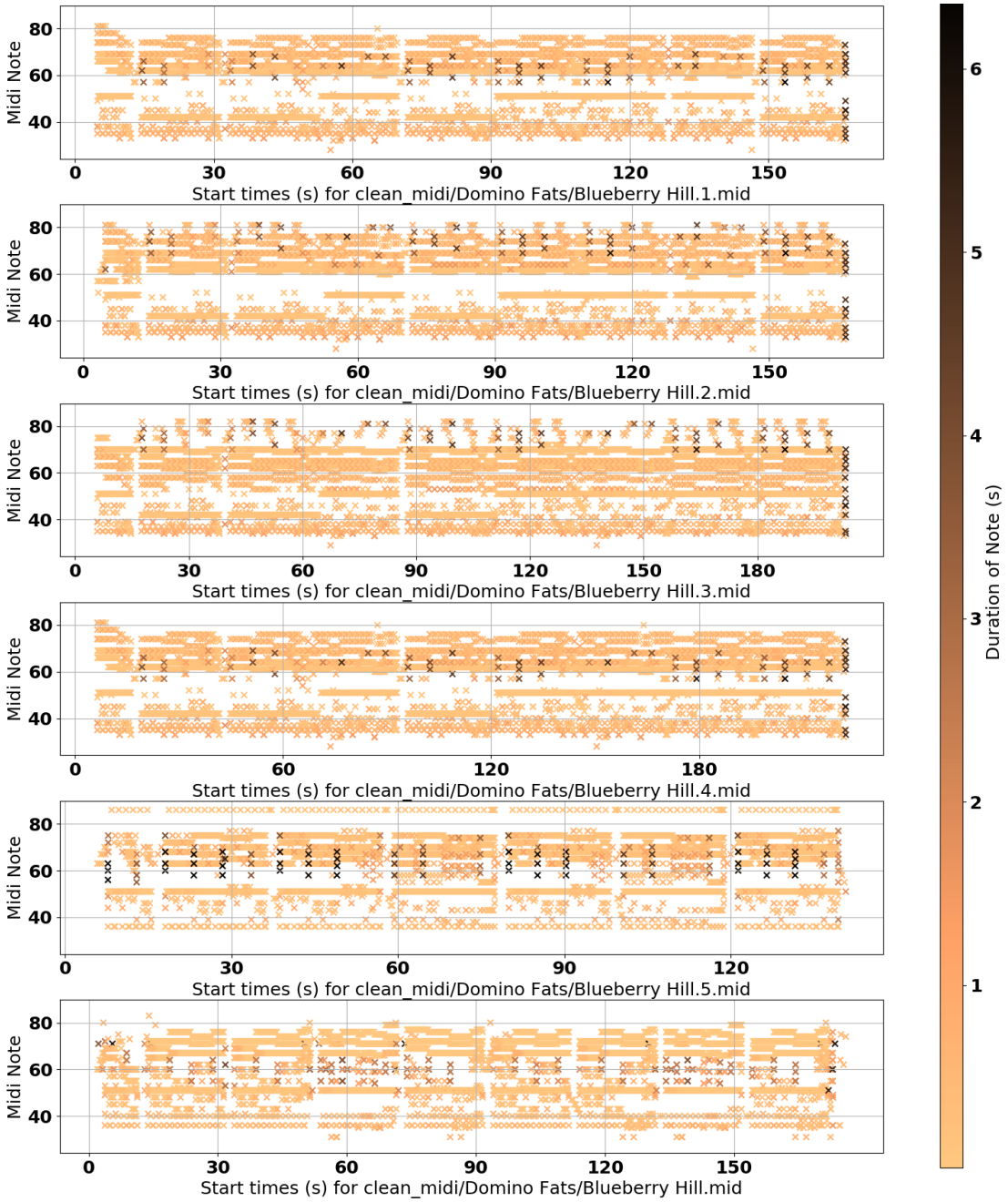


Figure 2.6: Spectrograms of files named "Blueberry Hill"

2.3 NOTE NORMALIZATION

Looking at the spectrograms and seeing similarities was helpful, but some were in different keys and it's difficult to tell what key a song is in by looking at the spectrogram. In Figure 2.6, one can see that the first, second, and fourth look very similar, the third is slightly similar to those, but the last two look completely different. Are they still the same song?

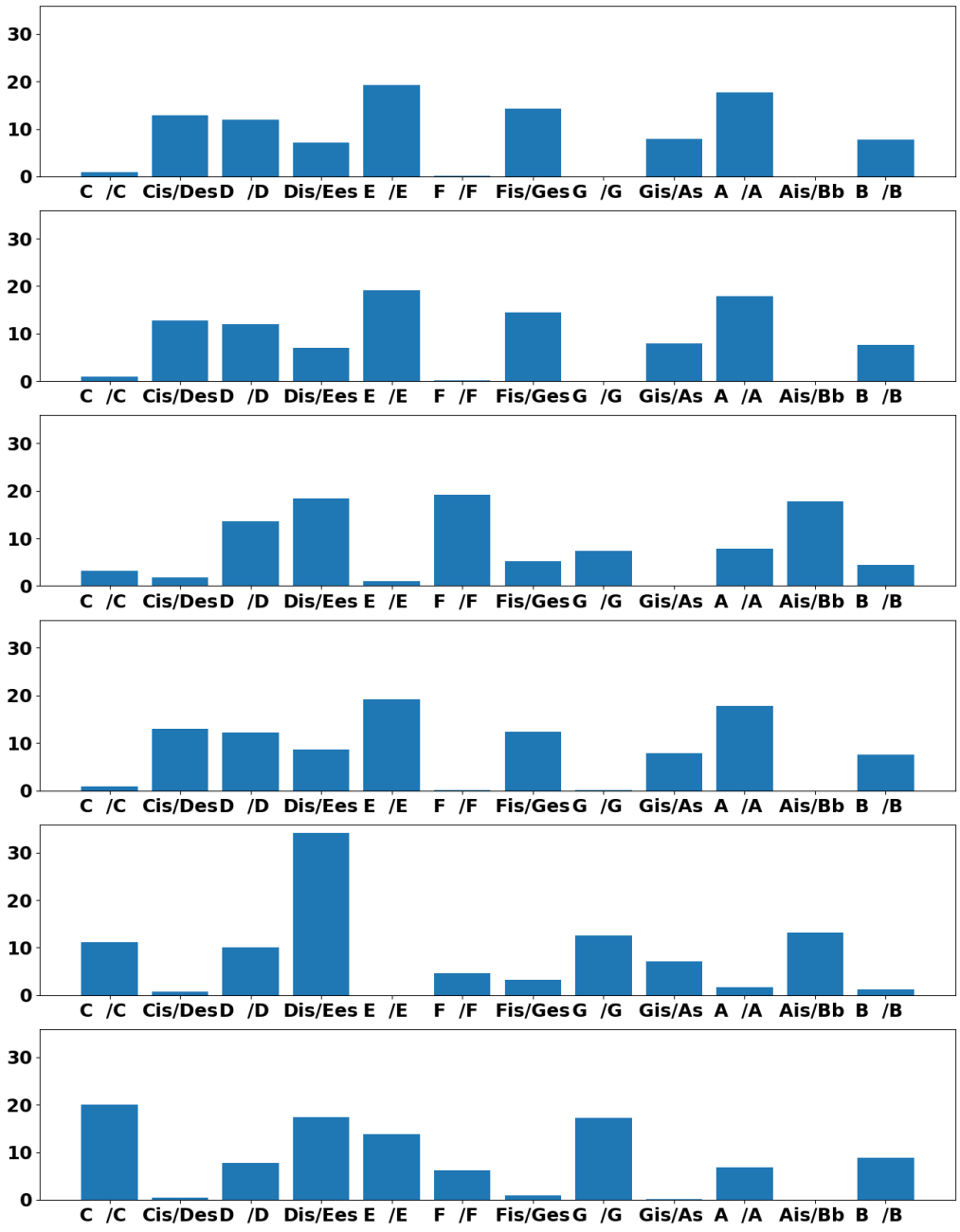


Figure 2.7: Histograms of Notes for "Blueberry Hill" by number of notes.

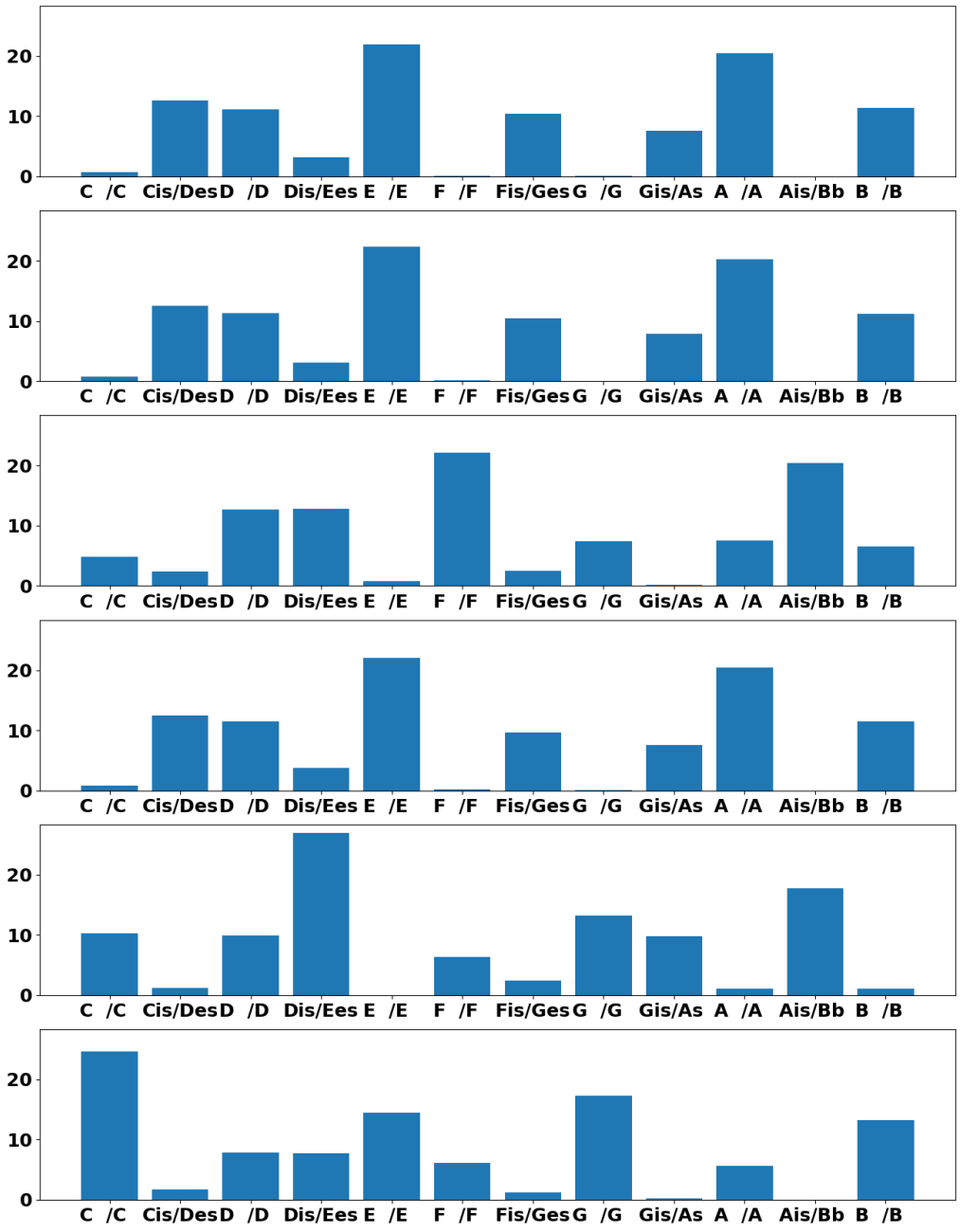


Figure 2.8: Histograms of Notes for "Blueberry Hill" weighted by duration.

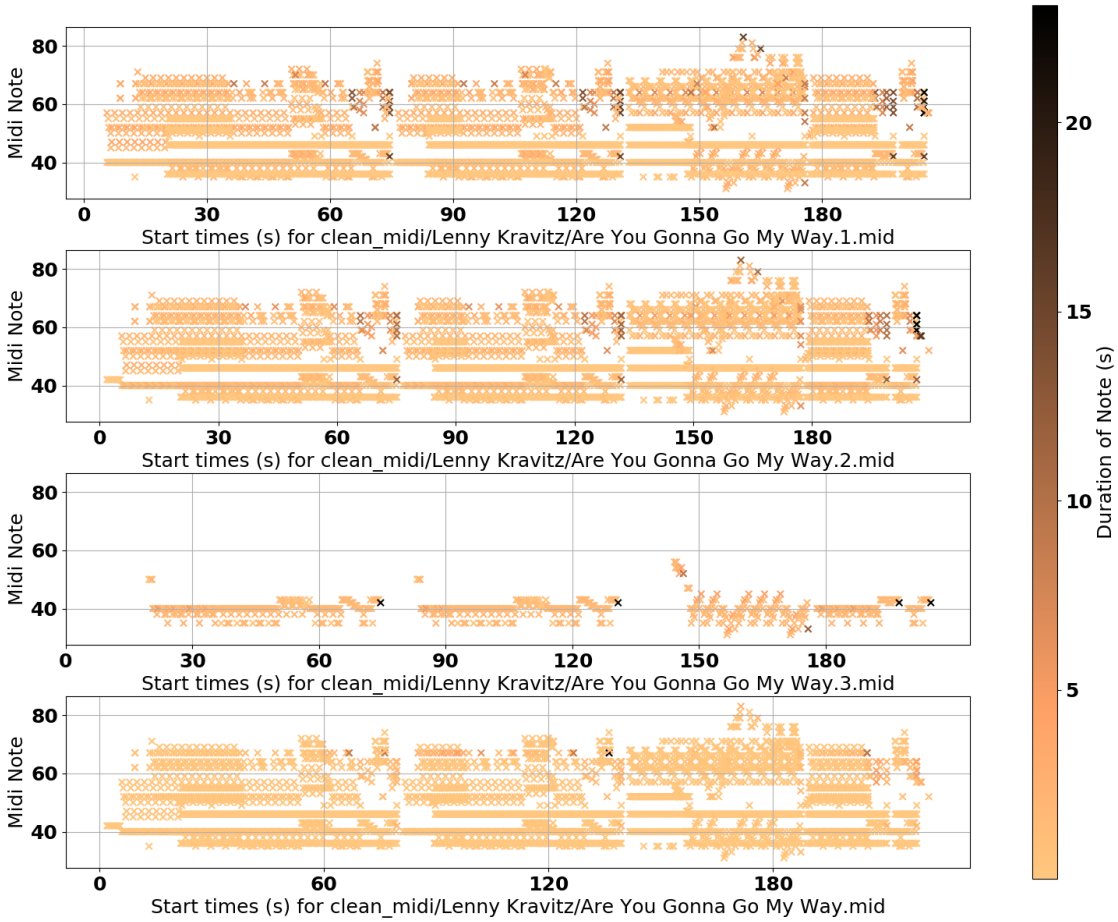


Figure 2.9: Spectrograms of files named "Are You Gonna Go My Way"

I decided to take a histogram of the named notes in each file of the song, using the number of times a note showed up in the song, or note count, in Figure 2.7 and weighting by duration in Figure 2.8.

2.3.1 Named Note Histograms

Figure 2.9 is what the spectrograms of four files for the same song look like. Note that the third is bass only.

Despite the differences that can be seen in the spectrogram in Figure 2.9, you can see that the histograms in Figures Figure 2.10 and 2.11 look strikingly similar. That makes sense because all four files are in the same key, and are of the same song.

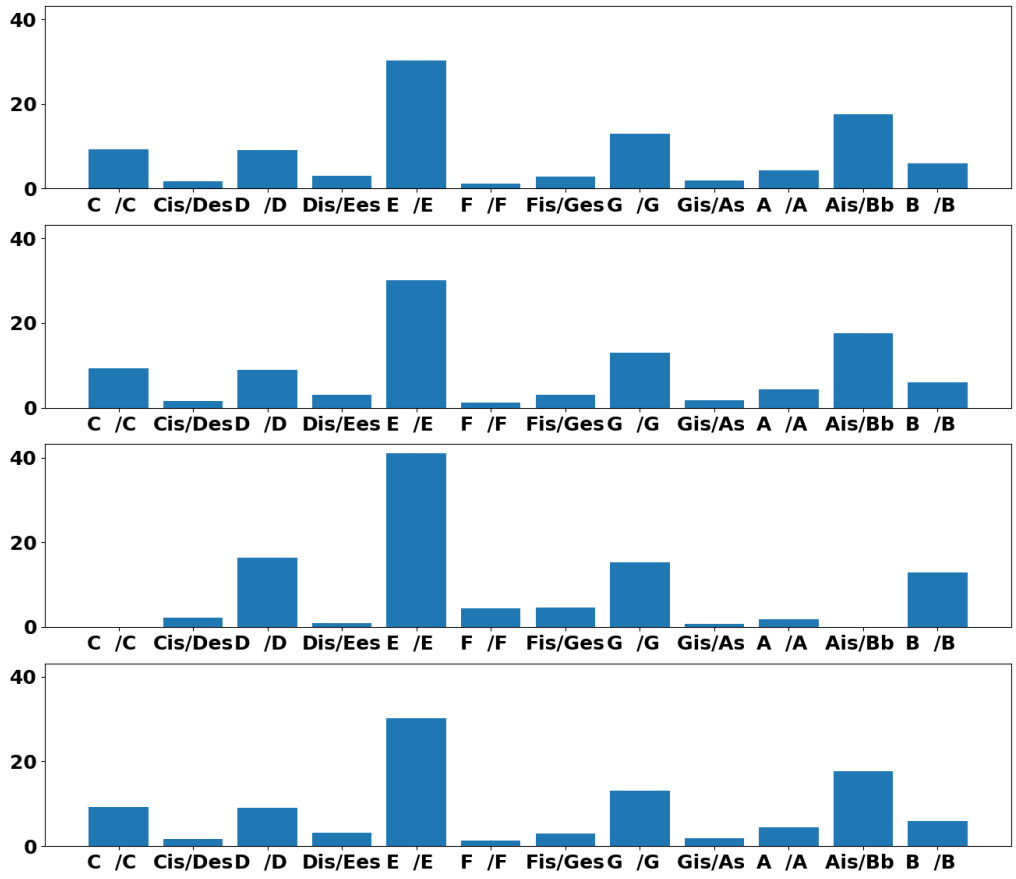


Figure 2.10: Histogram of Notes for "Are You Gonna Go My Way" by number of notes.

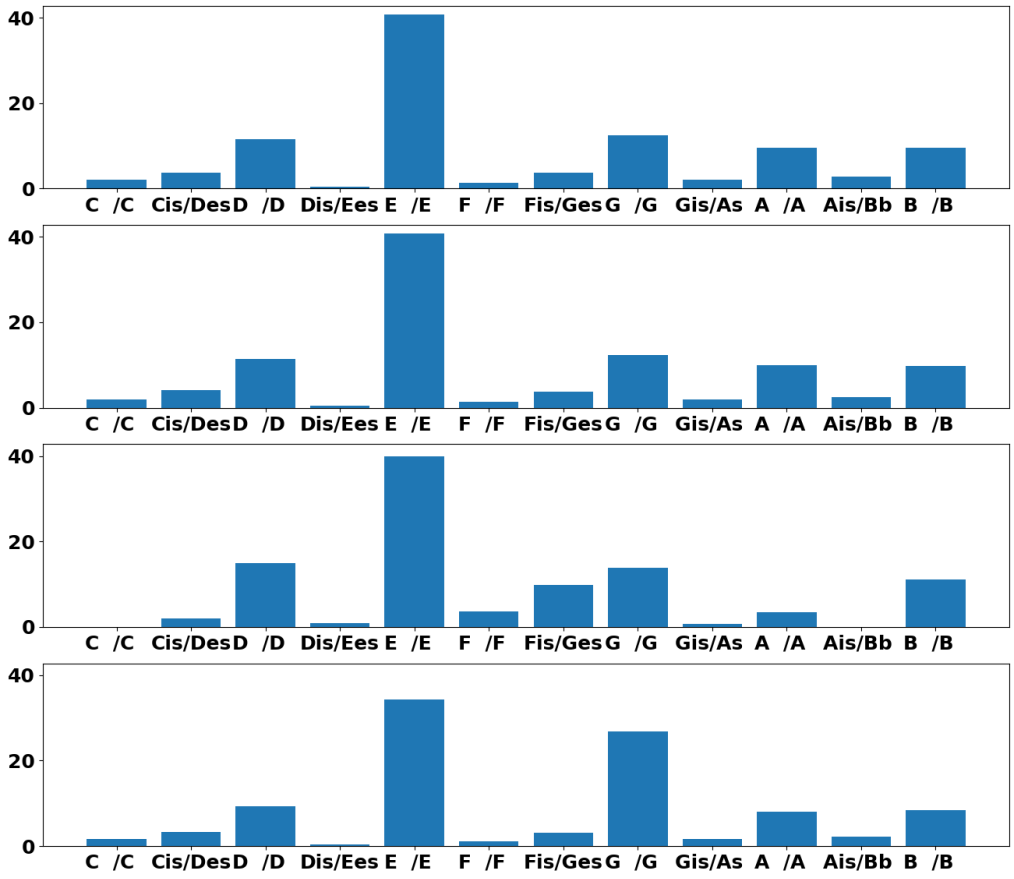


Figure 2.11: Histogram of Notes for "Are You Gonna Go My Way" weighted by duration.

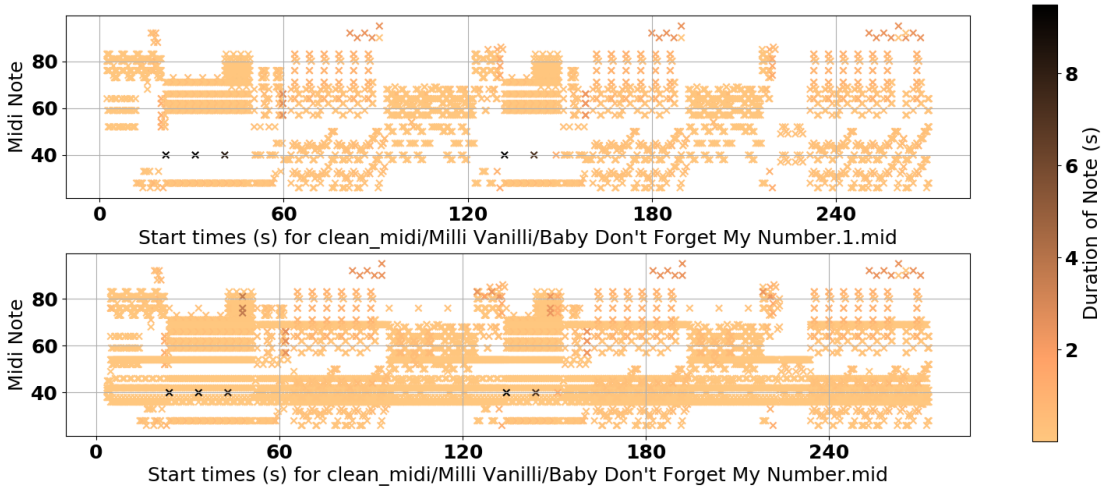


Figure 2.12: Spectrogram of files named "Baby Don't Forget My Number"

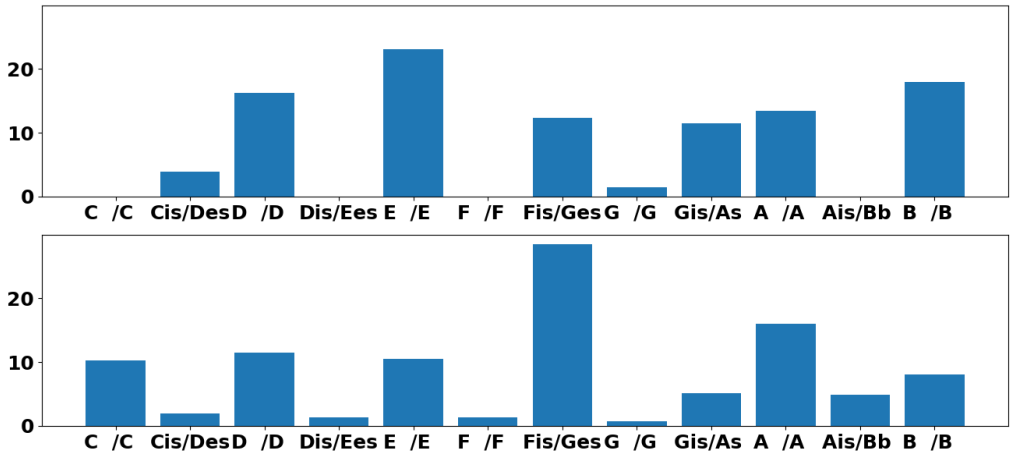


Figure 2.13: Histogram of Notes for "Baby Don't Forget My Number" by number of notes.

2.3.2 Weighted by Duration

Sometimes the histograms won't show similarities at all unless they're weighted by duration. The spectrograms shown in Figure 2.12 are very similar looking and are of the same song, but their histogram in Figure 2.13 doesn't really match. Weighting by duration in Figure 2.14, however, shows a clear match.

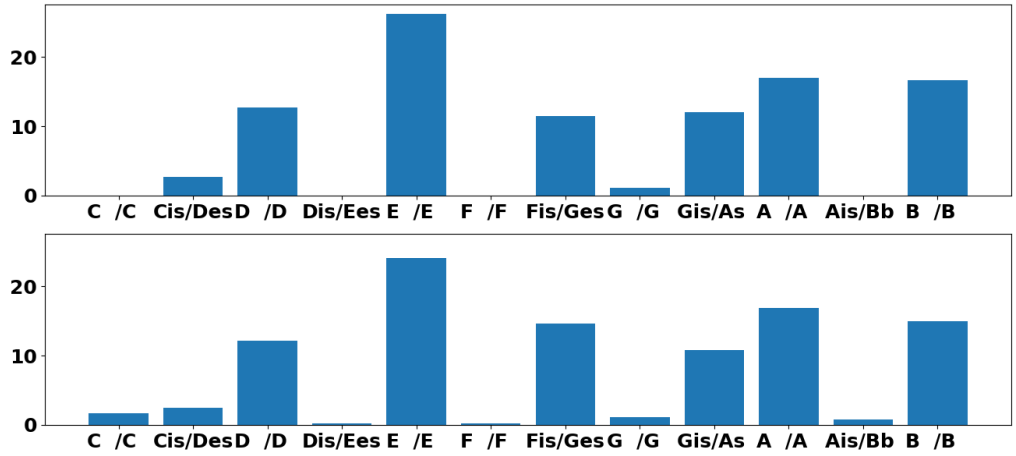


Figure 2.14: Histogram of Notes for "Baby Don't Forget My Number" weighted by duration.

This makes sense when we think about the way notes are used in a song. Some shorter notes that may not affect the melody include ornamentation and travel notes. Travel notes are not the primary notes of a song. They are typically very short and provide interest, which drives their note count high. But since their duration is very short, weighting by duration brings out the more important notes of the song.

Why did the note count work for "Are You Gonna Go My Way" even without weighting by duration? Here, we must also take into consideration the effect popularly seen in harpsichord music. The harpsichord is an instrument constructed similarly to a piano except, instead of hammers and dampers in a piano, it uses a plucking motion that renders a very short note. Due to the lack of sustain, harpsichord pieces were and are usually written with a very regular note pattern, repeating the important notes many times. The mandolin, a similar instrument, also has negligible sustain which produces similar results.

2.3.3 Key Changes

The histograms can also show key changes. Figure 2.15 shows another match that looks like the same song; however, the histograms in Figures 2.16 and 2.17 don't show an obvious match even when weighted by duration. It turns out that the song is shifted by three half-notes.

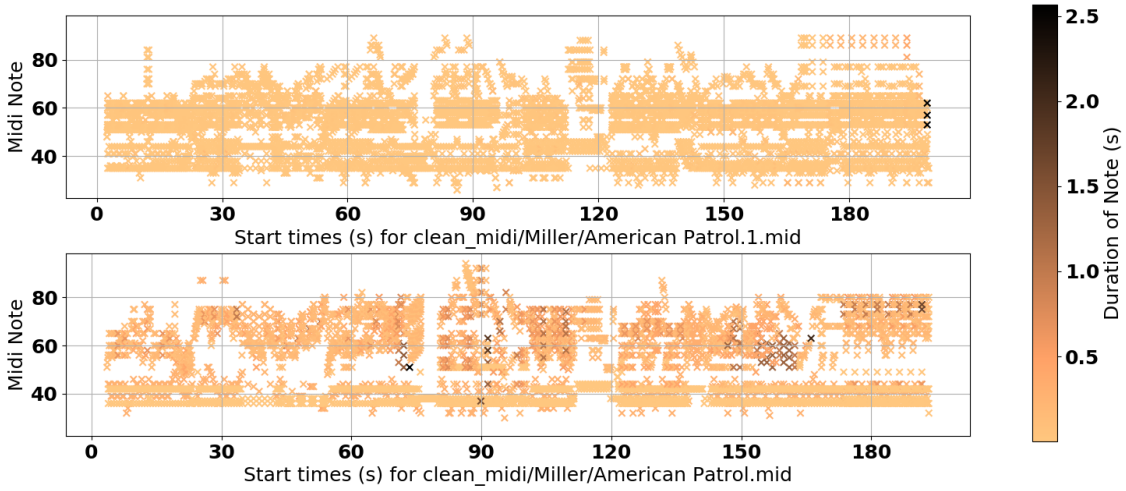


Figure 2.15: Spectrograms of files named "American Patrol"

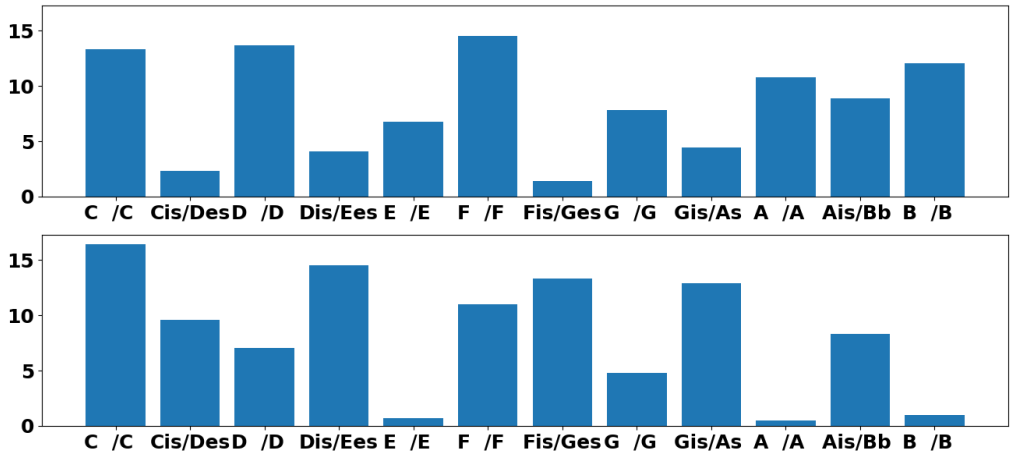


Figure 2.16: Histogram of Notes for "American Patrol" by number of notes.

In order to see the shift, move the entire bottom plot 3 notes to the left. The large peak in the bottom plot at Dis/Ees will match up with the large peak in the top plot at C/C; the dip in the bottom plot at E/E will match up with the dip in the top plot at Cis/Des; the large peak in the bottom plot at F/F will match up with the large peak in the top plot at D/D. This continues all the way until we wrap the left three notes in the bottom plot at C/C, Cis/Des, and D/D with the right three notes in the top plot at A/A, Ais/Bb, B/B. This shift is shown in Figure 2.18 using a hue spectrum. By matching the hues, we can see that the peaks and valleys match on the histogram.

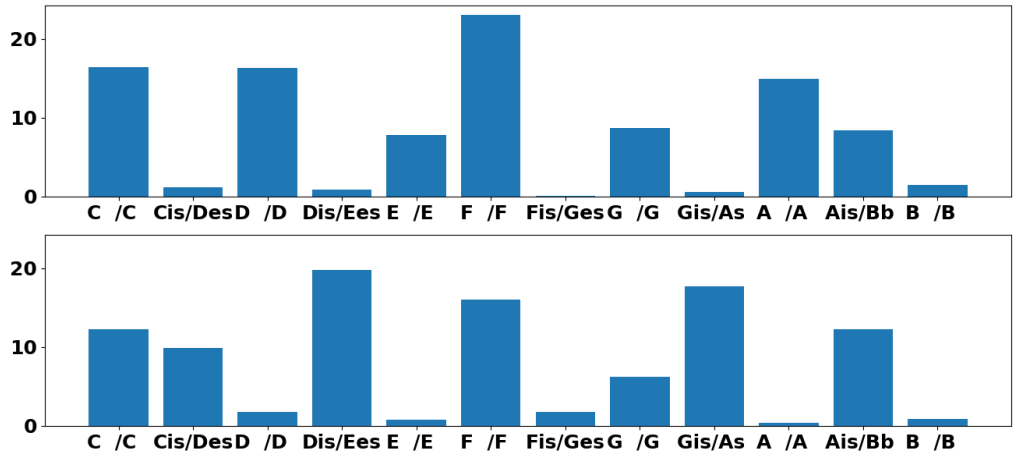


Figure 2.17: Histogram of Notes for "American Patrol" weighted by duration.

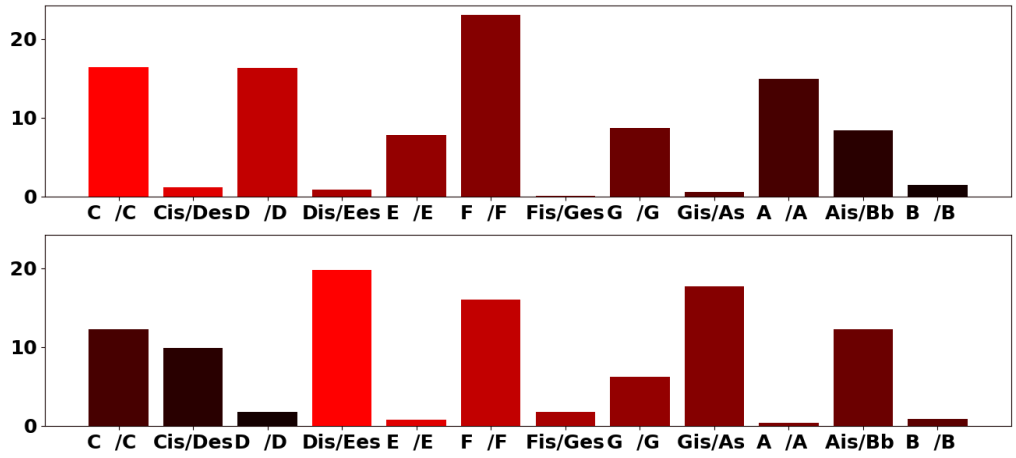


Figure 2.18: Pitch shift demonstration for "American Patrol" weighted by duration.

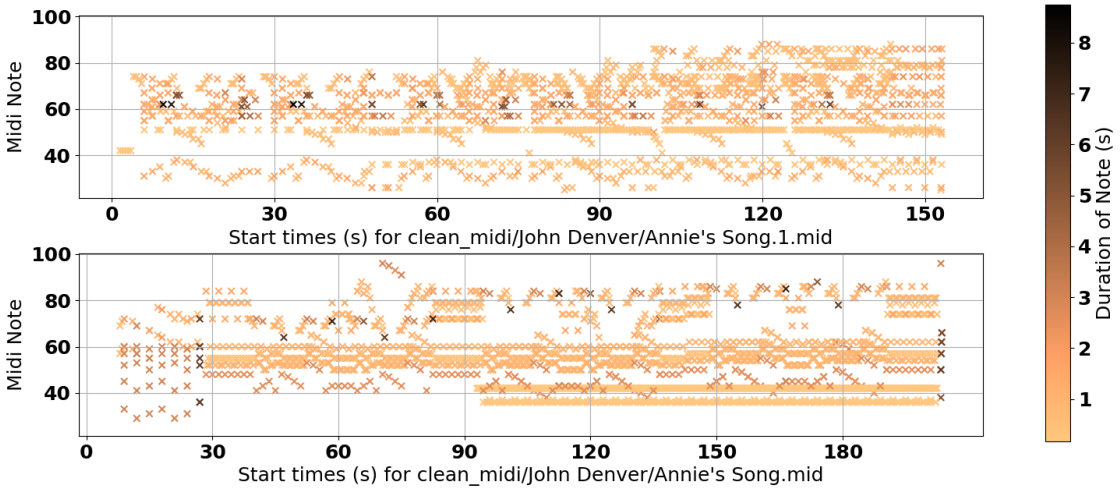


Figure 2.19: Spectrograms of files named "Annie's Song"

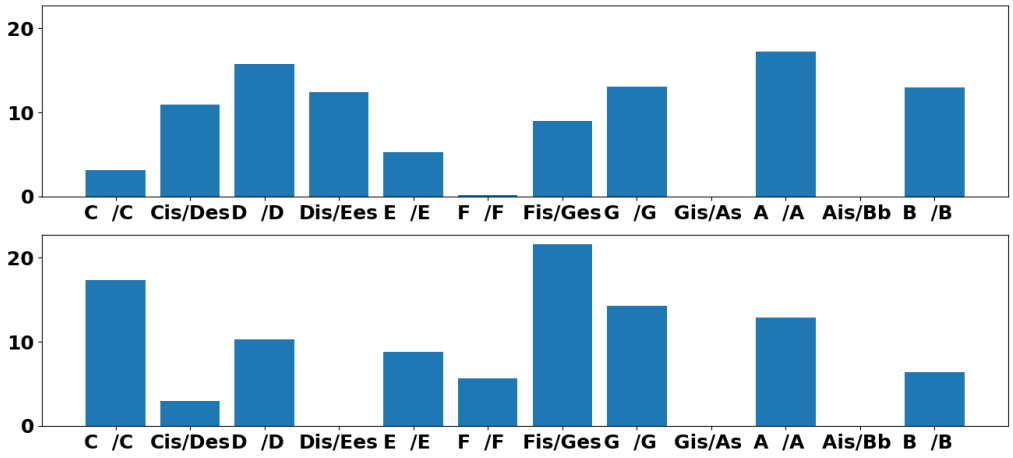


Figure 2.20: Histograms of files named "Annie's Song" by number of notes.

2.3.4 Annie's Song

Sometimes, however, the note histograms will not show everything. Annie's Song, shown in Figure 2.19, is an example of how two renditions of the same song can look completely different, both by note count in Figure 2.20 and by duration in Figure 2.21. I converted these two MIDI files into WAV files, and they are indeed the same song. However, the second file shown has a different introduction section and changes key in the middle of the song. There were several songs that changed key in the data set that I used. This is a good reminder that music is a creative art, after all.

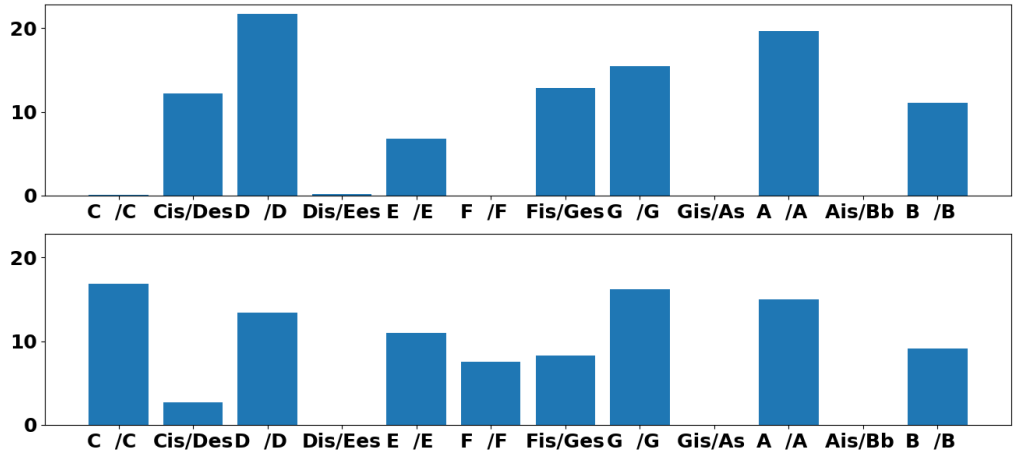


Figure 2.21: Histograms of files named "Annie's Song" weighted by duration.

2.3.5 Note Normalization

The histograms showed me that key normalization may not be as simple as choosing the top three notes. The nuance shown in the histograms is important in determining what matches. Since this becomes a pattern matching problem for the general shape of a 12–point histogram, this may be a good problem to throw at a machine learning algorithm to match against major, minor, and other musically standard scales to determine or normalize a song's key.

If nothing else, it can serve as an additional feature to be used in reliability or speed of the search. For reliability, one can get a histogram and run them against results to double check a match. For speed, one can search first by 12 transpositions of the histogram and use the best preliminary matches to search faster through the smaller dataset.

CHAPTER 3: CONCLUSION

Although Shazam is able to recognize exact tracks even in case of background noise, the algorithm is not fundamentally built to recognize musical structure. Differences in key and tempo easily confuse it, but there may be ways to adapt the Shazam algorithm to recognize them.

Despite my initial expectations, selecting simply by note duration, note velocity, and top three most popular named notes did not yield a very clear or consistent match for files of the same song. However, key detection or normalization can be done using histograms of the main named notes.

Tempo, ornamentation, and orchestration might be addressed by doing more preprocessing or additional calculation based on the structure of existing songs. See my thoughts about these in Section 3.1.

3.1 FUTURE WORK

Future work will include addressing the issues caused by key changes within the same song and determining criteria for choosing secondary points that form the edges in the graph.

For addressing issues caused by key changes within the same song, I mentioned windowing the song to check snippets instead of the entire file. Windowing is chopping the song into pieces of pre-determined length, and evaluating them individually. Some variables include length and overlap of the windows. Due to unintended side effects of windowing, one must also consider the tail effects, especially in tapering the windows and how they affect the requirements for overlap.

For choosing boxes to determine the second point in each pair, I have two questions. What kind of pitch and time limitation will produce the best fan factor? What magnitude of fan factor can a moderately large server system tolerate to serve the intended number of target audience? I might consider the note density of the pieces themselves and the general tendency in music to not jump more than an octave at a time. I might consider using student cloud subscriptions to test and tweak these parameters.

Beyond key changes, music recognition can also benefit from being tempo agnostic. If there was a way to pick out ornamentation and harmonization from the songs, it may help; however, the histograms may have already addressed that issue. I would welcome the opportunity to test this theory. Paying more attention to musical structure and variation during music recognition is distinct from the signal processing of specific tracks, and I am excited to see that there is a lot more to discover about it.

REFERENCES

[1] A. L.-C. Wang, "An industrial-strength audio search algorithm," in *Proceedings of the 4th International Conference on Music Information Retrieval*, 2003.

[2] SoundHound, "Soundhound," 2018. [Online]. Available: https://soundhound.com/

[3] Midomi, "Midomi," 2009. [Online]. Available: https://www.midomi.com/

[4] G. P. M. Help, "Identify songs playing near you," 2018. [Online]. Available: https://support.google.com/googleplaymusic/answer/2913276?hl=en

[5] Huang, Kim, Hasegawa-Johnson, and Smaragdis, "Singing-voice separation from monaural recordings using deep recurrent neural networks," 2014. [Online]. Available: http://paris.cs.illinois.edu/pubs/huang-ismir2014.pdf

[6] Unknown, "The rights of man," 2018. [Online]. Available: https://thesession.org/tunes/83

[7] J. C. B. Paris Smaragdis, "Non-negative matrix factorization for polyphonic music transcription," 2003. [Online]. Available: https://www.ee.columbia.edu/ dpwe/e6820/papers/SmarB03-nmf.pdf

[8] A. A. Matthias Dorfer and G. Widmer, "Towards end-to-end audio-sheet-music retrieval," 2016. [Online]. Available: https://arxiv.org/pdf/1612.05070.pdf

[9] S. Sigtia, E. Benetos, and S. Dixon, "An end-to-end neural network for polyphonic piano music transcription," 2016. [Online]. Available: https://arxiv.org/pdf/1508.01774.pdf

[10] V. Sarnatskyi, V. Ovcharenko, M. Tkachenko, S. Stirenko, Y. Gordienko, and A. Rojbi, "Music transcription by deep learning with data and artificial semantic augmentation," 2017. [Online]. Available: https://arxiv.org/pdf/1712.03228.pdf

[11] J. LEE, K. YOON, D. JANG, S.-J. JANG, S. SHIN, and J.-H. KIM, "Music recommendation system based on genre distance and user preference classification," 2005. [Online]. Available: http://www.jatit.org/volumes/Vol96No5/11Vol96No5.pdf

[12] J. Despois, "Finding the genre of a song with deep learninga.i. odyssey part. 1," 2016. [Online]. Available: https://hackernoon.com/finding-the-genre-of-a-song-with-deep-learning-da8f59a61194

[13] S. J. Haitsma and T. Kalker, "A highly robust audio fingerprinting system," 2002. [Online]. Available: http://www.ismir2002.ismir.net/proceedings/02-FP04-2.pdf

[14] D. Ellis, "Robust landmark-based audio fingerprinting," 2009. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/23332-robust-landmark-based-audio-fingerprinting

[15] Musipedia, "Musipedia," 2018. [Online]. Available: https://www.musipedia.org/

[16] F. Xiao and S. Oderberg, "Converting audio to midi with neural networks," 2017, Final Project Report.

[17] C. Raffel, "The lakh midi dataset v0.1," 2016. [Online]. Available: https://colinraffel.com/projects/lmd/

[18] C. Raffel, "Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching," Ph.D. dissertation, Columbia University, 2016.

[19] gin66, "midi2ly," May 2016. [Online]. Available: https://github.com/gin66/midi2ly

[20] fangfx, "midi2ly," Aug. 2018. [Online]. Available: https://github.com/fangfx/midi2ly