Inferring pre-defined Hierarchical Wave Function Collapse constraints from MIDI files

Chaan van den Oudenhoven
Supervisor(s): Rafa Bidara
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering June 23,
2024

Name of the student: Chaan van den Oudenhoven
Final project course: CSE3000 Research Project
Thesis committee: Rafa Bidara, Joana de Pinho Gonçalves

Inferring pre-defined Hierarchical Wave Function Collapse constraints from MIDI files

Chaan van den Oudenhoven
Delft University of Technology
Delft, The Netherlands

ABSTRACT

AI-generated music is a huge research field with many different approaches and models being the result of it. One such model is the ProceduraLiszt model, which utilizes the Wave Function Collapse algorithm, an algorithm similar to constraint programming, to generate its music. This research builds upon that model. It does so by trying to reverse engineer a given piece of music into a set of satisfied constraints that the model is compatible with. We present an approach that allows for the inference of constraints of a given music file that adheres to the MIDI format called MidiAnalyser. We run our model on a set of music files and analyze the inferred constraints. The constraints include aspects like key and note range.

CCS CONCEPTS

• **Applied computing** → **Sound and music computing**; • **Theory of computation** → **Constraint and logic programming**.

KEYWORDS

wave function collapse, procedural music generation, constraint programming, music information retrieval

ACM Reference Format:
Chaan van den Oudenhoven. 2024. Inferring pre-defined Hierarchical Wave Function Collapse constraints from MIDI files. In *Proceedings of Research Project (CSE3000).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Music is a widely popular art form that can be appreciated by practically anyone, even those who have no talent for making music, and yet it allows for a near-infinite amount of nuance. this, however, can make it difficult to define properly what music is. There are of course plenty of ways to write music down, music sheets have been around for thousands of years but this only goes for individual pieces of music. How, for example, would one define the features of a certain "style" or "genre" of music? And more importantly, how would you extract these features from a piece of music if given an example? Wave Function Collapse (WFC) is an algorithm that has proven to be very promising when it comes to procedurally generating music. WFC works in a similar way as constraint programming by splitting the result that has to be produced (in this case a piece of music) into a bunch of variables and subsequently defining constraints between neighbouring ones. Initially, every variable is a superstate in which it can take every value of its original domain, but as neighbouring variables have their domain reduced, it eventually collapses into a single value. The constraints determine which combination of variable values are allowed and they usually represent the problem that has to be solved.

this however begs the question: if we can generate music using constraints, could we then take a given piece of music and retrieve the constraints? What constraints can we find in different pieces of music? Are there any patterns? The aim of this research is to analyze different pieces of music to see if there are any patterns in the we can find in the constraints that they adhere to.

CSE3000, June 23, 2024, Delft

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 RELATED WORK

2.1 Music information retrieval

Extracting features from music pieces is a field of science that has been extensively studied and is referred to as the field of Music Information Retrieval (MIR). Ryan Stables et al [1] for example introduces the "SAFE" system which allows for the extraction of semantical descriptions of musical timbre within a given piece of music.

2.2 Applying Constraint programming to generating music

Additionally, there have been multiple studies done on the potential constraint programming has when it comes to procedurally generating music. Constraint programming is a paradigm used for solving combinatorial problems. it does so by defining the problem space as a set of variables and subsequently applying constraints to subsets of those variables. These constraints prohibit the affected variables from taking on certain combinations of values. Constraint programming can be applied to music generation by translating fundamental music rules into sets of constraints. Notably, Anders, T. and Miranda, E. R, [2] present a formal model for producing convincing chord root progression by defining constraints between chord pitch class sets and roots.

2.3 MIDI

All of the musical analysis that was done throughout this paper was performed on Musical Instrument Digital Interface (MIDI) files as our model was, at the time of our experiments, not compatible with any other music formats.

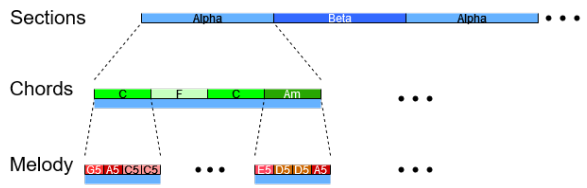

Figure 1: Basic HWFC model for procedural music generation, proposed by Varga and Bidarra [4].

MIDI files are a standard format used for communicating and storing musical performance data. Instead of storing raw audio like most music storage formats, they instead store a series of messages that dictate how music should be played. These messages include notes, velocity, tempo, and control signals.

2.4 Wave Function Collapse and Hierarchical WFC

Wave Function Collapse (WFC) is an algorithm that has shown a lot of promise when it comes to the field of Procedural Content Generation (PCG). Its inspiration comes from the field of quantum mechanics and the concept of wave functions. Wave functions represent the infinite amount of states that a quantum particle can have as one superstate using a probability function. Eventually, the superposition collapses and the particle takes on a singular value. The original WFC algorithm was conceptualized by Maxim Gumin [3]. The algorithm operates by representing whatever has to be generated as a grid of cells. A common example would be the generation of a digital image, which WFC would represent as a grid of pixels. Subsequently, a set of constraints are applied to subsets of the grid space. These constraints are usually derived from a given input and they limit the combinations of values that the cells can take. If the model was given an image of an empty chessboard, it would for example derive the constraint that tiles cannot neighbor tiles of the same color (as is the case on a chessboard). The algorithm collapses each of the cells into a value until either every cell is collapsed or it discovers that there is no combination of values that satisfies all constraints.

3 RELEVANT MUSIC THEORY

Music is a form of art that is constructed using sound. It consists of multiple elements that work together to create its composition. The primary building blocks that make up music are known as measures, chords, and notes.

- Notes: Notes are the fundamental pieces that make up even the simplest of music. They represent the distinct sound that is being played along with its duration. Each note also has a pitch, which represents how high or low the note sounds.
- Chords: Chords are combinations of notes that are played simultaneously. They are responsible for the overall harmony of the song. Usually, the notes that make up a chord are of a lower pitch than the ones that make up the melody as the chords are intended to serve as a sort of background noise.
- Measures (or bars): Measures are a unit of time music uses. They are primarily responsible for ensuring that the song has a consistent tempo by organizing the notes and chords within them. The time signature of the song determines how many beats are in one measure. Practically every song can be divided into a certain amount of measures.

4 REPRESENTING MUSIC

The musical structure of a song can vary greatly. Be it in terms of beats per minute, chord progressions, or overall melody. Nevertheless, to properly infer a set of constraints from a piece of music, we need some kind of universal representation that we can transform a given piece into. That way we can define a set of general constraints that we can check on any piece. If we were to use multiple representations, than inferring a specific constraint in one piece could mean something else in another.
Varga and Bidarra [4] introduced a model for procedural music generation, based on Wave Function Collapse. Instead of a single canvas, this model features three levels (sections, chords and melody), with each cell on the upper levels being associated with an entire canvas of cells on the level below (see Figure 1). The level below is formed from the concatenation of all of these canvases, with each one belonging to a single cell on a higher level, but containing multiple cells. The specific layers are as follows:

- Sections: at the top level, the overarching structure of the piece is laid out. This is where, for example, an intro section is placed before the first verse, or the bridge is followed by one last chorus.
- for each section, a canvas of chords is created, each of its slots representing one chord. For the sake of simplicity, we can think of the duration of one chord as one measure in the piece.
- at the lowest level, for each measure, a canvas of notes is available, which will give us a short segment of the melody.

Constraints can be specified by the composer, either by directly filtering the domain, or by drawing up certain relations that neighboring cells should obey. Some parameters of some constraints may depend on values chosen for the cells above in the hierarchy. So-called prototypes can also be defined, and when these are chosen for a value of a given cell, the canvases underneath this cell in the hierarchy may get new constraints, the ones specified with the prototype.
The main benefit that this representation of music provides is that it allows us to separate the individual notes of a song into a set of cells while still being able to conserve information about the relation between notes. This representation achieves this through the notion of having notes be neighbors of other notes and by having them be a part of the same chord or section. This is useful for the purposes of our research because it allows us to infer constraints that span multiple notes by simply checking the notes that belong to the same higher-level cell. For these reasons, we decided that our approach would be based on this representation of music.

5 TRANSLATING MUSIC INTO OUR REPRESENTATION

Creating music is a process that can have a large amount of variance. It requires the composer to determine things like the beats per minute the song will use, the chord progression, or how long

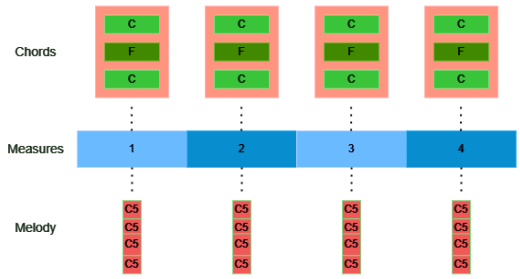

Figure 2: Structure used by the MidiAnalyser to model arbitrary piece of music

the song will be, among other things. This variance, however, can make it quite tricky to translate an arbitrary piece of music into our own music representation.
We had to modify some of the assumptions used by the original music generation model. For example, in the original model, we structure our music so that it consists of a set of sections, which in turn consist of chords, which in turn have an associated melody. In an arbitrary piece of music, however, we cannot assume that the song adheres to this hierarchical structure. For example, it is common for there to be parts of a song during which no chords are being played. These parts would then only consist of a set of notes that make up the melody. Alternatively, a song might have parts with multiple chord changes happening within the same measure. In such a case, the melody of the measure would play during multiple different chords.
Due to this variance in music compositions, we had to make some moderate changes to the way we represented music. Instead of positioning chords as higher-level cells and melodies as lower-level cells, we disassociate the two and instead group them by the measure during which they are played. The way this is structured can be seen in Figure 2. In this new structure, we can still represent songs that solely consist of a melody, without the need to accommodate for the missing chords.

6 METHODOLOGY

6.1 MidiAnalyser

We introduce the MidiAnalyser model. This model uses a slightly different representation of music compared to the one that the original ProceduraLiszt uses. In the original model, we define a hierarchical structure in which we assume that every chord in a piece of music has a set of notes that are played at the same time. This chord and the accompanying notes then make up what we would call a measure. When dealing with arbitrary music pieces, however, this is an assumption we cannot make. Often times music pieces will have sections in which there simply aren't any chords, but only a melody. Or alternatively, you might encounter pieces with multiple chords in a single measure.
After we successfully divided the music into chords and notes we ran our model. For each section or note we checked for each constraint if the part satisfied the constraint. The following constraints were the ones used for the chords:

- Chord in key constraint: Checks if the chord is in a given musical key. A musical key is a set of notes that usually form the basis of a piece of music.

The following constraints were used for the melodies:

- melody in range constraint: checks if all of the notes are within a certain range of pitches (for example: are all of the notes of this melody in between C5 and C6?).
- descending melody constraint: checks if the pitches in the melody are descending.
- ascending melody constraint: checks if the pitches in the melody are ascending.

Some of the variables require a given variable to be checked, for example, the MelodyInRangeConstraint requires two input notes to check if the entire melody is within those notes. In cases like these, we simply check what the strictest variables are that the music part adheres to as that implies that the part also adheres to more relaxed MelodyInRangeConstraints.
The research was mainly performed on MIDI files from the NES music repository [5]. The advantage of using these songs is that they have relatively simple melodies and chords alongside usually having a single instrument.

6.2 Music21

Running the constraint inferrer requires the given piece of music to be in a format that is compatible with our hierarchical model. This however requires us to isolate notes and chords from the music piece. While MIDI files list the individual notes that make up the music, chords are only represented if they are manually annotated in the file, something that is very rarely the case. For this reason, we use the Python package "Music21" [6]. Music21 allows us to parse a given MIDI file and convert it into a Stream. A Stream in this context refers to the way Music21 represents music. Streams essentially act as containers that can hold things like music parts and measures. The measures in turn then contain notes and chords that we can use for our model.

6.3 ProceduraLiszt

The ProceduraLiszt model presented by Varga and Bidarra in which the MidiAnalyser is implemented comes with a fully functional GUI. To perform the analysis, it requires the following input

- A music file encoded in the MIDI (Musical Instrument Digital Interface) format
- The selected pre-defined constraints that the user wishes to infer from the music file.

upon performing the analysis the model outputs the following:

- For each measure, the selected constraints that the chord and melody within that measure fulfill.
- A visual representation of the Notes. The different chords and melody parts are made visible through the use of coloring.

7 EXPERIMENTAL WORK

We first run the MidiAnalyser on pieces of music generated by the Proceduraliszt model. We do this to verify that the constraints found by our model are valid. Since we know which constraints were used to generate the music, we can check if our model finds

the same constraints when we give the generated music as input. we generate three songs using the Proceduraliszt model with varying parameters to showcase that our model is capable of handling music of different kinds:

- The first song will have an equal number of sections, chords, and melody length along with a melody with pitches in the range of C5 to C6.
- The second song will have an equal amount of sections and chords, but varying melody lengths throughout the measures

The first song we generated was done so with the following parameters:

- Bpm: 120
- Number of sections: 4
- Number of Chords: 4
- Melody length: 4
- Constraints: melody in key constraint (C Major), melody in range (C5 to C6), ascending melody

Parameters for the second generated song:

- Bpm: 90
- Number of sections: 4
- Number of Chords: 3
- Melody length: 3
- Constraints: melody in key constraint (D minor), melody in range (C5 to C6), melody starts on note (C)

To clarify what this means, generating a song using the model with the parameters of the first song will result in a song with 4 sections, each consisting of 4 chords. Each chord in turn has a melody consisting of 4 notes. Considering that the original model considers each measure to contain one chord, this will result in a song with 16 measures.
After this, we ran the MidiAnalyser on various songs from the NES music library to discover what constraints we could find. We chose this library because of the relative simplicity of the songs within it since these songs tend to use a singular instrument. We ran the MidiAnalyser on the following songs from the library:

- `037_BoobyKids_09_10EvilDante.mid`
- `071_DigitalDevilStory_MegamiTensei_17_18LastBattleLastBoss.mid`
- `018_ArumananoKiseki_00_01Opening.mid`

8 RESULTS

8.1 Generated Music

When running the MidiAnalyser on the music that was generated by the WFC model, we see that it correctly identified all of the constraints that were used to generate the music. In Figure 3 we see that it detected that the melody of every measure was in the correct key of C Major and it detected that all of the melody notes were in the range of C5 to C6. Similarly in Figure 4, we see that the MidiAnalyser identifies that the melodies are in the key of D Minor, the melody range being in C5 to C6, and that every melody started with the note C#. To showcase the results of the MidiAnalyser when run with constraints that it should fail, we analyzed our second generated song twice, but this time with the chord in key constraint of C4 to C5. Subsequently, none of the measures are in that range as is depicted in Figure 5

8.2 NES music files

We ran the MidiAnalyser on various music files from the NES music Library. The following examples are meant to showcase various constraint patterns our model is capable of finding.

- Booby Kids Evil Dante: The results are depicted in Figure 6. According to our model, we find that the melody of every measure in the song falls in the range of C5 to C6 and we also discover that most of the melody is in the key of C Major. We also see that some of the measures have an ascending melody while others have a descending melody. It is possible for a measure to adhere to both of these at the same time. This generally means that the entire melody consists of the same note.
- Digital Devil Story Megami Tensei Last Battle Last Boss: The results are depicted in Figure 7. According to our model, we find that the melody of every measure in the song falls in the range of C3 to C6 and we also discover that measures 9 through 22 have a melody that is in the key of C Major. Additionally, we see that some of the measures have an ascending melody while others have a descending melody, although a noticeable pattern is difficult to discern.
- ArumananoKisekiOpening: The results are depicted in Figure 8. According to our model, we find a repeating pattern when it comes to the key of the melodies. We see the (albeit short) pattern of the first measure not being in C major while the following two are.

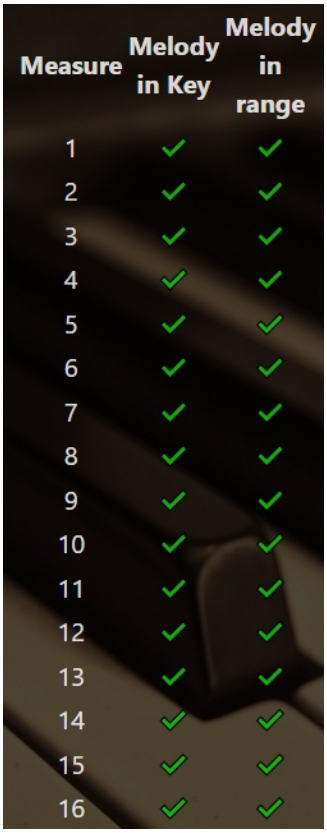

Figure 3: MidiAnalyser results from the first generated song shown in the interface of the constraint analyzer within the ProceduraLiszt app. The x-axis depicts the different constraints we selected while the y-axis depicts all the measures of the song

Figure 4: MidiAnalyser results from the second generated song shown in the interface of the constraint analyzer within the ProceduraLiszt app. The x-axis depicts the different constraints we selected while the y-axis depicts all the measures of the song

Figure 5: MidiAnalyser results from the second generated song shown in the interface of the constraint analyzer within the ProceduraLiszt app. This time the analyzed melody in range constraint is set to C4 to C5 while the song was generated with C5 to C6. The x-axis depicts the different constraints we selected while the y-axis depicts all the measures of the song

Figure 6: MidiAnalyser results from BoobyKidsEvilDante shown in the interface of the constraint analyzer within the ProceduraLiszt app. The x-axis depicts the different constraints we selected while the y-axis depicts all the measures of the song. The melodyKey is C Major and MelodyRange is C5 to C6

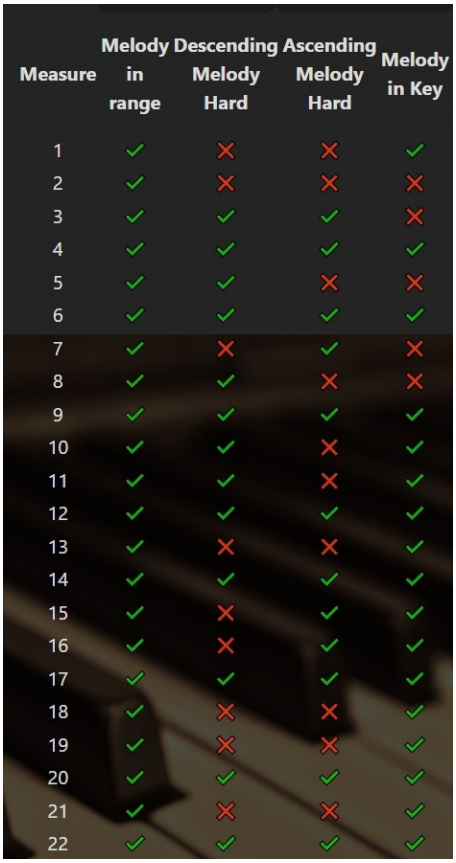

Figure 7: MidiAnalyser results from DigitalDevilStoryMegamiTenseiLastBattleLastBoss shown in the interface of the constraint analyzer within the ProceduraLiszt app. The x-axis depicts the different constraints we selected while the y-axis depicts all the measures of the song. The melodyKey is C Major and MelodyRange is C3 to C6

Figure 8: MidiAnalyser results from ArumananoKisekiOpening shown in the interface of the constraint analyzer within the ProceduraLiszt app. The x-axis depicts the different constraints we selected while the y-axis depicts all the measures of the song. The melodyKey is C Major and MelodyRange is C5 to C6

9 DISCUSSION

9.1 Limitations

This paper has showcased the potential our model has when it comes to analyzing various constraints that songs can adhere to. However, both due to time constraints and this research being done by one person, WE had to limit the scope of this research. For this reason, there are still several limitations to our analysis that should be discussed. Improvements upon these limitations could serve as promising future research.

- When it comes to analyzing songs, the model does not perform well on music that is too complex. Particularly, songs that have multiple instruments with different pitch ranges can make it quite difficult to separate the melody notes from the chord notes. Currently, the model uses an external library to perform this task, so future research should focus on using more advanced music separation techniques.
- While not completely relevant to the research topic of this paper, there are considerable improvements that could be made to the GUI that the model uses. Most of the current elements are essentially prototypes and were not made with Human Computer Interface (HCI) principles in mind. Future research regarding this topic should seek to improve upon this and make the MidiAnalyser more accessible.
- Several constraints that the WFC model uses to generate its music were not compatible with the music representation that our model uses. There is for example "MelodyStartsOnNote" constraint for example, which allows the user to generate music where each measure starts on a specific note. This constraint allows the user to choose a custom note, but in the WFC model, it also allows the user to have the melody start on the root of the chord the melody belonged to. Due to the MidiAnalyser no longer using a hierarchical structure, constraints that assume a relation between the chords and melody of a measure are not compatible. Future research should aim to produce a model that allows for the possibility of relations between the melody and chords of a song.

10 CONCLUSION

In this paper, we have explored the challenge of implementing a model that is capable of inferring pre-defined constraints from various Midi music files. We have highlighted the music properties that had to be accounted for when translating music files into our representation, with the major one being the lack of a hierarchical structure when it comes to music in real life. We proposed a method for analyzing music along with a new representation that can be used to model arbitrary music. Subsequently, we analyzed a variety of music pieces using our model and showcased the various constraints that we were able to infer. The analyzed music pieces included both pieces generated by the WFC model that our model is based on and pieces acquired from the NES music library Overall, our research contributes to a deeper understanding to the inferring of constraints in various music files allowing composers to possibly detect constraint patterns in music that they or others create.

11 RESPONSIBLE RESEARCH

Transparency and Explainability: Transparency and explainability are crucial in the deployment of AI technologies like WFC, especially in creative domains such as music composition. For WFC, transparency means that users can understand how the algorithm makes decisions, which patterns it chooses, and how it interprets constraints. Explainability involves providing insights into why certain choices are made by the algorithm—such as why a particular chord progression or melody was generated—allowing users to trust and effectively interact with the technology. This is particularly important in collaborative environments where artists may wish to tweak algorithmic suggestions.

WFC is a white-box algorithm, meaning that given an input, anyone could generate an output, solely based on the description of the algorithm. This property is preserved in the hierarchical model used for music generation.

Enhancement vs. Replacement: The mixed-initiative approach of WFC in music generation underscores its role as a tool for enhancing rather than replacing human creativity. In this setup, there is an iterative loop between the user and the algorithm. The process starts with the user defining initial constraints, which the WFC algorithm then uses to generate a composition that adheres to these parameters. Upon reviewing the algorithm's output, the user can draw inspiration and identify aspects they might want to alter, leading them to tweak the input constraints based on their creative judgment and preferences. This iterative process allows for a deeply interactive and collaborative form of composition, where the algorithm serves as both a source of inspiration and a sophisticated tool that extends the creative capacities of the user. By enabling a creative dialogue between the composer and the algorithm, WFC supports the artistic process, allowing composers to explore new ideas and refine their work, all while ensuring that the technology remains a helpful tool rather than taking over the creative process.

Use of Data: In the application of WFC for music generation, the algorithm operates uniquely as it does not require any training data or examples to function. Instead, it relies entirely on explicit constraints and music pieces provided directly by the user. This approach significantly shifts the data use paradigm in AI by eliminating the need for large datasets and the associated concerns of data privacy and copyright infringement. By relying on user-defined constraints and music, WFC allows composers to maintain full control over the creative output, ensuring that the compositions are both original and closely aligned with the artist's intent.

REFERENCES

[1] Ryan Stables et al. Safe: A system for the extraction and retrieval of semantic audio descriptors. Master's thesis, Birmingham City University, 2014.
[2] T. Anders and Miranda. A computational model that generalises schoenberg's guidelines for favourable chord progressions. Master's thesis, University of Plymouth, 2009.
[3] Maxim Gumin. Wave function collapse algorithm. 2016.
[4] Pál Patrik Varga and Rafael Bidarra. Harmony in hierarchy: Mixed-initiative music composition inspired by wfc, 2024. Submitted for publication.
[5] andJulianMcAuley ChrisDonahue, HuanruHenryMao. Thenesmusic database:amulti-instrumentaldatasetwithexpressiveperformanceattributes, 2018.
[6] Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data, 2010.