May 3rd , 2021

Music Accompaniment Generator

Dhruva Panyam

Nishant Mahesh

Contents

1. Introduction

Creativity is something that is heavily attributed to humans, and their ability to generate new ideas, and alter existing ones. A human endeavour that is often associated with being creative is the creation of art - be it in the form of sculptures, literature, paintings, architecture, photography or *music*.

Noam Chomsky, a pioneer in the field of psychology and linguistics, described humans' ability to construct an infinite number of sentences for communication as a *creative process* [2].
This project is interested in exploring ways to define a finite grammar, that is capable of simulating human creativity, specifically in the context of music, and chord accompaniment generation.

The approach that this project adopts to try and achieve this goal, is to design a Context Free Grammar (CFG), that will be used by a web application to generate chords that accompany a user-inputted melody, while also giving the user the option to choose the *style* in which accompaniments must be generated.

The following sections will give a brief description of the music theory that a reader will require to understand this report, followed by a detailed description of how the CFG was designed, and how it can then be used to generate the required musical accompaniments.

2. A Brief on Music Theory

2.1 Notes in Western Music

In Western music, there are a total of 12 notes, namely **C, C#, D, D#, E, F, F#, G, G#, A, A#, B** (after which the sequence keeps repeating). We call the interval size between any 2 successive notes in this sequence a *semitone*. Similarly, an interval size of 12 semitones is called an *octave*.

Each of these notes are used to represent an equivalence of pitches or frequencies. For example, the **A** note represents the equivalence class of frequencies:

$$\{A_0 = 27.50Hz, A_1 = 55.0Hz, A_2 = 110Hz, A_3 = 220Hz, A_4 = 440Hz...\}$$

2.2 Music Scales

A *music scale* is defined as a specific sub-sequence of the 12 notes, typically identified by the starting note and the interval size between successive notes.

For example, the **C Major scale** comprises of the following notes:

{C, D, E, F, G, A, B, C} ***Listen***

Similarly, the **C Natural Minor scale** comprises of the following notes:

{C, D, D#, F, G, G#, A#, C} ***Listen***

A song or piece of music is usually written using notes of a particular scale. This scale is called the *key signature* of the song. A song can also include a few other notes not belonging to its key signature, which are called *accidentals*. However, for the purpose of simplicity, this project ignores accidentals for the most part, and assumes that songs are made up of just the notes of their scales.

2.3 Melodies and Chords

2.3.1 A Melody

Now that we've defined what notes and scales are, a *melody* is simply some sequence of notes, belonging to a particular scale, played one after the other. These notes are often sung by the lead singer, or played by the lead instrumentalist.

Again, for the purpose of simplicity, we ignore the *rhythmic* aspect of these melodies, and assume that these individual notes are played in series, at regular intervals of time.

Melody Example

Melodies however, only constitute one half of a song – the other half that accompanies these melodies are called *chords*.

The same melody along with accompaniment

2.3.2 A Chord

A chord can be defined as a specific group of notes (typically 3 - 4 notes) played simultaneously. Similar to melodies, the notes of each chord for a given song is governed by the key signature of the song. These chords are played alongside the notes of a melody, forming a complete piece of music.

Chord Examples
(In the audio, each chord is preceded by the individual notes that make up the chord)

Much like we uniquely identified different scales using their starting note and the intervals between different notes, a chord too can be identified by its **root note** (the note in the group with the lowest pitch) and the intervallic relationship between other notes (which determines the **chord type**).

2.3.3 Chord Types

A chord in its simplest form, is typically made up of 3 notes – called a *triad*. Depending on the intervals between each note of the triad and the root note, these chord types different names.

A **major triad** for example, consists of the **root note**, **a 4-semitone interval** (formally called a *major third* interval) and a **7-semitone interval** (formally called a *perfect fifth* interval).

Similarly, a **minor triad** consists of the **root note**, a **3-semitone interval** (called the *minor third* interval) and a **7-semitone interval** (the perfect fifth interval).

The **C Major** and **C minor** chords are simply made of the **major** and **minor triads** respectively, rooted at the **C** note (i.e),

C Major $\equiv$ **C, E, G** played simultaneously.
C minor $\equiv$ **C, D#, G** played simultaneously.
Listen to C Major followed by C minor

In this project, we will identify each chord uniquely by its **root note** and its **chord type**. Thus, the **C Major (C-M)** has root **C** and type **Major (M)**, while the **E minor (E-m)** chord has root **E** and type **minor (m)**.

Adding on to this basic structure, more complex chords can be formed by adding more notes to this group of 3 notes. To give a few examples:

- Adding a **11-semitone interval** (formally, a *major seventh* interval or the **B** note) to the **C Major** triad, gives us a **C-Major-7** chord (**C-M7**), with notes **C, E, G, B**.
- Adding a **10-semitone interval** (formally, *minor seventh* interval or the **A#** note) to the **C minor** triad, gives us a **C-minor-7** chord (**C-m7**), with notes **C, D#, G, A#**.

A more comprehensive list of all the chord types that are relevant to this project can be found in Table 3.2.

2.3.4 Relationship Between Chord and Melody

At this point, we have only mentioned that a piece of music constitutes a melody, and a group of chords which accompanies this melody. This section will explain exactly how this accompaniment is chosen, i.e, which chords can accompany which melody notes.

Although in reality the relationship between melody notes and chords may be more nuanced, a simple rule that governs chord-melody relationships is as follows:

A chord can accompany a given melody note, if the chord consists of that melody note.

Thus, for the purposes of this project, **C Major (C-M)** would be a valid chord accompaniment for the **G** note, while **F Major (F-M)** would not (since **F-M** consists of only **F, A, C**).

2.4 Styles of Accompaniment

This section will talk about how the choice of chords that are picked to accompany a given melody can influence the *feel* or *style* of the piece of music.

Broadly speaking, the 3 key factors that characterise the style of a song are:

- The **scale** of the song.
- The **chords** and **chord types** used.
- The **likeliness** of one chord following another.

Some examples of styles that we will deal with in this project are **happy, sad** and **jazz**.

A **happy** song is typically written in a **major scale**, where most chords are of the **Major (M)** chord type, while a **sad** song is typically written in a **minor scale**, with more chords of the **minor (m)** chord type.

Finally, depending on the style of the song, the likeliness of a particular chord immediately following another chord varies. This change from one chord to the next is called a *transition*. Consider a happy song written in the **C Major scale**. For this song, the **C-M** chord is associated with a sense of "home", since the **C** note is the root note of the scale. The **G-M** chord, on the other hand, is associated with a feeling of being "away from home". So, the **G-M** chord is highly likely to transition to the **C-M** chord, since chords have a tendency to resolve back to "home".

3. Formal Notations

In this chapter we will standardise the notations that we will henceforth use in this report to refer to all that has been described in Chapter 2.

3.1 Symbols for Melody Notes

The 12 notes in an octave {**C, C#, D, D#, E, F...,B**} will be denoted by the numbers `{0, 1, 2, 3,.., 11}`, with the number `0` always referring to the **C** note (See Table 3.1).

Note	*C*	*C#*	*D*	*D#*	*E*	*F*	*F#*	*G*	*G#*	*A*	*A#*	*B*
Number	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*

Table 3.1: *A table of notes and their numeric notations for this project*

3.2 Symbols for Chord Types and Their Expansions

Each chord type (**M, m M7, m7** etc.), will be represented as a list of numbers, with each number corresponding to a note of the chord type, where the root note is **C** (i.e, the number `0`).

For example, the **Major** chord type (represented by the symbol **M**) will be described as: `[0, 4, 7]`.
The **minor seventh** chord type (represented by the symbol **m7**) will be described as: `[0, 3, 7, 10]`.

A list of all chord types, along with the standardised notation and explicit chord expansion can be found in Table 3.2.
The chord type symbols that have been used in this project were adapted from the notations used in the *Mark Granroth-Wilding Jazz Corpus* [1]

3.3 Chord Naming Convention

As mentioned in Section 2.3.3, we will uniquely identify each chord using its **root note** and **chord type**, (i.e) using the following symbol: <*root_note*> - <*chord_type*>.

For example, the **F Major** chord will be represented using the symbol: **5-M**, and the **D minor seventh** chord will be represented using the symbol: **2-m7**.
It naturally follows, that the explicit expansion of a chord named **<r> - <t>** with root note **<r>** and chord type **<t>** will be:

$$[< note_1 > + < r >, < note_2 > + < r >, < note_3 > + < r >, ...] \text{ (mod 12)}$$

where $< note_i >$ is the i^{th} note in the chord expansion of the chord type **<t>**, according to Table 3.2.

Formal Name	**Our Notation**	**Chord Expansion**
minor	*m*	*0, 3, 7*
Major	*M*	*0, 4, 7*
minor seventh	*m7*	*0, 3, 7, 10*
Major seventh	*M7*	*0, 4, 7, 11*
dominant seventh	*7*	*0, 4, 7, 10*
augmented seventh	*aug7*	*0, 4, 8, 10*
diminished seventh	*o7*	*0, 3, 6, 9*
dominant seventh, flat five	*b5,7*	*0, 4, 6, 10*
dominant seventh, suspended fourth	*sus4,7*	*0, 5, 7, 10*
half diminished	*%7*	*0, 3, 6, 10*
minor seventh, sharp five	*#5,m7*	*0, 3, 8, 10*

Table 3.2 *A table of formal chord type names according to music theory, along with our standardised notation and chord expansion*

4. Designing a Context Free Grammar

This chapter will describe the design of the context free grammar that will be used by our application to generate accompaniments for melodies of a fixed length. It will describe the structure of the grammar (the terminals, variable names, rules etc.), the language of the grammar, and how the grammar can be used to extract chord accompaniments for a melody by using a parse tree.

4.1 Language of the Grammar

The input melody for the grammar is a sequence of 16 notes. These 16 notes are preceded by the style that the accompaniment is desired in. So, the format is:

$$< style >< note_1 >< note_2 > ... < note_{16} >$$

As mentioned in earlier sections, each *style* of a piece of music has an associated *scale*. Since a scale is a set of notes from [0-11], a valid melody for a style, must only consist of notes from the style's scale. In addition to this restriction, we make explicit rules for each style, which are explained further in Section 4.3.2.

Thus, it follows that the language of the grammar is the set of strings: $< style >< note_1 >< note_2 > ... < note_{16} >$ such that all the notes belong to the scale of $< style >$, and follow the explicit rules of that style.

4.2 Terminals

From the previous section, we can see that the terminals for the grammar are the different style names, along with the numbers [0-11] which represent different notes (see section 3.1). These are:

{happy, sad, jazz_major, jazz_minor, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

4.3 Structure of the Grammar

Below is the first part of the grammar, which derives the style of the accompaniment.

U → Happy | Sad | Jazz_Major | Jazz_Minor

Happy → *happy* (Start_H)

Sad → *sad* (Start_S)

Jazz_Major → *jazz_major* (Start_JM)

Jazz_Minor → *jazz_minor* (Start_Jm)

Definitions:

U: *Start symbol*

Happy, Sad, Jazz_Major, Jazz_Minor: *Variables for each style*

happy, sad, jazz_major, jazz_minor: *Terminals for each style*

Start_X: *Beginning of melody notes for style with short-form as X*

4.3.1 Variables

The variables observed so far are the start symbols, the style variables, and the melody start variables for each style. Each style's start variable has the following rules:

Start_X → (B1_X) (B2_X) (B3_X) (B4_X)

Each of B1, B2, B3, and B4 for a style X represent a *bar*, which is a group of 4 notes. This structure exists mainly for readability. Other variables include the following: **one for each chord name** (<r> - <t> format) across all styles and **one global chord variable for each style**, Any_X, which is a variable that can yield all chord names that belong to style X.

4.3.2 Styles and Chord Types

Each style has a set of chords associated with it. For instance, the **happy** style has the following chords: 0-M, 2-m, 4-m, 5-M, 7-M, 9-m. For our application, we have set in place certain explicit rules for each style. These rules say that the first and last notes of the melody must be derived from a specific chord that represents "home" (Section 2.4). For instance, a melody's first and last notes for the **happy** style must be derived from the 0-M chord.

These explicit rules are implemented as follows:

Start_H → (B1_H) (B2_H) (B3_H) (B4_H)

B1_H → (0-M) (Any_H) (Any_H) (Any_H)

B2_H → (Any_H) (Any_H) (Any_H) (Any_H)

B3_H → (Any_H) (Any_H) (Any_H) (Any_H)

B4_H → (Any_H) (Any_H) (Any_H) (0-M)

4.3.3 Chord Rules

We know that each chord name has two components: the **root** and the **type**. For example, the root of `5-m7` is `5`, and the type is `m7`. For each chord type, we know the list of notes it represents, starting at `0`. In this case, the type `m7` has the notes `[0,3,7,10]` (see Table 3.2). When combined with the root as `5`, all the notes are shifted by 5 (modulo 12). So, we get `5-m7 = [5,8,0,3]`.

So, all the variables for the various chord names have rules that yield each note of their list. In the case of `5-m7`, it will have the following rules:

5-m7 → *5*

5-m7 → *8*

5-m7 → *0*

5-m7 → *3*

4.3.4 Complete Structure

To demonstrate a clear picture of the structure of the grammar, below is the structure for the style **happy**. The same structure is implemented for all other styles.

U → Happy | Sad | Jazz_Major | Jazz_Minor

H → happy (Start_H)

Start_H → (B1_H) (B2_H) (B3_H) (B4_H)

B1_H → (0-M) (Any_H) (Any_H) (Any_H)

B2_H → (Any_H) (Any_H) (Any_H) (Any_H)

B3_H → (Any_H) (Any_H) (Any_H) (Any_H)

B4_H → (Any_H) (Any_H) (Any_H) (0-M)

Any_H → 0–M | 2–m | 4–m | 5–M | 7–M | 9–m

0–M → *0* | *4* | *7*

2–m → *2* | *5* | *9*

4–m → *4* | *7* | *11*

5–M → *5* | *9* | *0*

7–M → *7* | *11* | *2*

9–m → *9* | *0* | *4*

5. Derivations

5.1 Justifying the Structure

It can be argued quite easily that this language of the context-free grammar (CFG) is a *regular* language. So, naturally, the question of why we use a CFG arises. While CFGs definitely assist in checking whether an input belongs to a language, they serve other purposes as well. Similar to evaluating arithmetic or logical expressions using CFGs, our application requires the structure of *parse trees*. We explain how the parse tree is utilized in further sections.

5.2 Derivation Procedure

To derive the parse tree for a given input, we implement the left-most derivation. This involves expanding the left-most variable at every step until there are only terminals. To carry out this process, we make use of the structure of our grammar. Since each note of the input melody is derived from one chord, we make note of this chord, and move on to the next variable. In order to optimize this process for repeated calculations, we make use of dynamic programming to keep track of partial derivations that have been made. The next section demonstrates this process using an example.

5.3 An Example

Consider the input string: `happy 0 0 7 7 9 9 7 7 5 5 4 4 2 2 0 0`
Fig. 5.1 shows a possible derivation for this input.

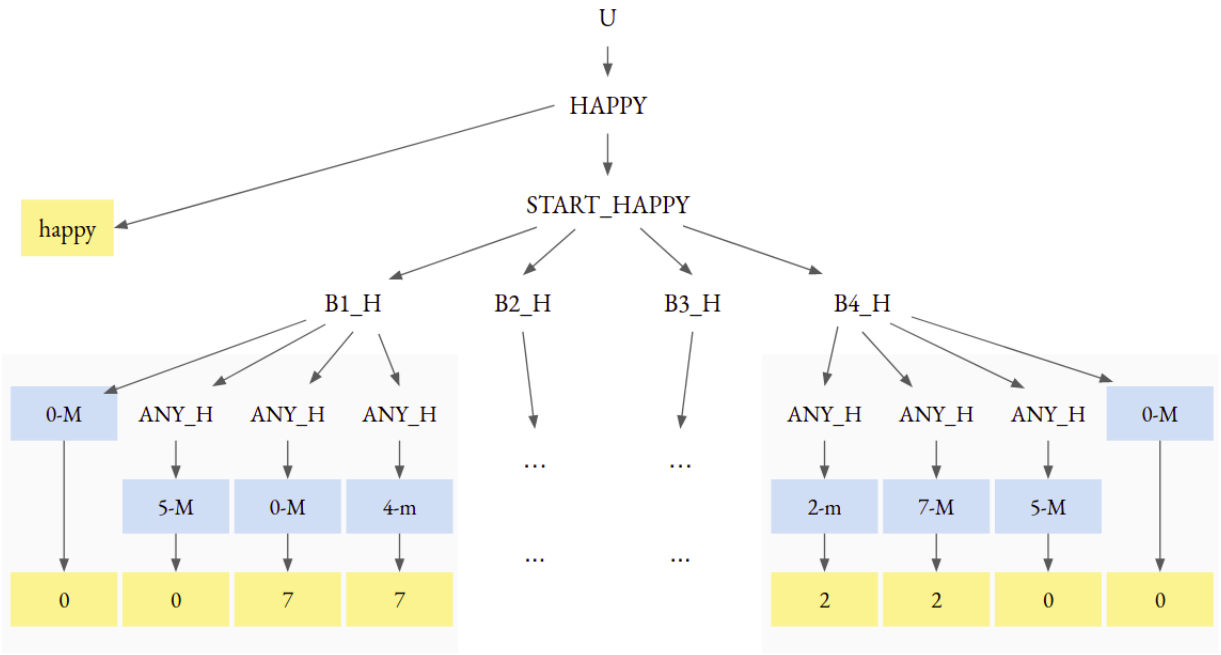

Fig 5.1: *An example of a parse tree for a given melody:*
"happy 0 0 7 7 9 9 7 7 5 5 4 4 2 2 0 0".
The yellow boxes contain all the terminals from the input.
The blue boxes are the desired accompaniment.

Once this parse tree has been successfully derived, we obtain the accompaniment for the melody by compiling all the chord names corresponding to each note of the melody (blue boxes).

5.4 Transition Probabilities

5.4.1 Need for Transition Weights

Section 2.4 mentions that one of the factors that helps characterise the style of a song is the likeliness of one chord being followed by another chord. While the actual music theory that influences this likeliness may be complex, we wanted a simple way to incorporate these probabilities while generating accompaniments.

We do this by creating a matrix of transition weights, where a "likeliness score" is assigned for each chord C_1 being followed by another chord C_2 in a song. (See Table 5.1 for an example)

5.4.2 Representation of Transition Weight Data

Each style's matrix contains a list of weights for each chord of the style. A sample list of weights for a chord in a style is given in Table 5.1.

The probability that chord C_2 follows chord C_1 in the style X can be derived as follows:

$$P(C_2|(C_1, X)) = \frac{weight[X][C_1][C_2]}{sum(weight[X][C_1])}$$

The manner in which these probabilities are used to influence the parse tree derivation is explained in the next section.

	0-M (C Major)	**2-m (D Minor)**	**4-m (E minor)**	**5-M (F Major)**	**7-M (G Major)**	**9-m (A minor)**
0-M (C Major)	*40*	*10*	*10*	*50*	*50*	*20*
2-m (D minor)	*50*	*10*	*10*	*40*	*50*	*10*
4-m (E minor)	*40*	*10*	*10*	*50*	*50*	*10*
5-M (F Major)	*50*	*10*	*10*	*50*	*40*	*10*
7-M (G Major)	*60*	*10*	*10*	*40*	*50*	*10*
9-m (A minor)	*50*	*10*	*10*	*40*	*40*	*10*

Table 5.1: *The weight matrix for the "happy" style, where each row represents chord C_1 and each column represents chord C_2.*

5.4.3 Integrating with Parse Tree Derivation

Parse Tree Generation Without Weight Matrix:

Without the chord transition weights, the left-most derivation is carried out as follows:

Let us assume that the input is: `happy 0 0 7 7 9 9 7 7 5 5 4 4 2 2 0 0`. At some stage of the parse tree derivation, the current branch may look like the following:

`Branch:` `happy 0 0 7 7 9 9 7 7 5 5 4 4 2 2 (Any_H) (0-M)`

`Input :` `happy 0 0 7 7 9 9 7 7 5 5 4 4 2 2 0 0`

We can ignore the first 15 terminals, and focus on the last two symbols. The branch is `(Any_H 0-M)`, and it must derive the string `"0 0"`. Since `Any_H` can yield all chords belonging to the style **happy**, any

chord that derives 0 is a valid yield for `Any_H`.

For instance, the following chords yield the terminal 0:

`0-M` → `0 | 4 | 7`

`5-M` → `5 | 9 | 0`

`9-m` → `9 | 0 | 4`

So, valid yields from the current branch are: `(Any_H) (0-M)` ⇒ `(0-M 0-M) | (5-M 0-M) | (9-m 0-M)`.

However, for our application, since we return the *first* valid derivation that we find, the derivation that is returned depends completely on the order in which the rules of the variables are expanded. Since our application revolves around the fact that the grammar is ambiguous, the only way to incorporate variety in the kind of chord accompaniment generated is by deriving different parse trees for the same input. To do so, we incorporate the transition weights to re-order the rules of certain variables.

Incorporating the Weight Matrix:

To modify the order of the rules, we must keep track of the previous chord at all times. So, once a terminal is reached successfully, the chord that yielded this terminal is stored, which will then influence the chord that comes next. Since the weights define *chord* transitions, we only need to reorder the rules of variables that can yield multiple chords. In our grammar, the only variables that yield multiple chords are `Any_H`, `Any_S`, `Any_JM`, and `Any_Jm` (see Section 4.3.4). When one of these variables are encountered during the left-most derivation, the rules of the variable are reordered using the weight matrix values of the previous chord.

Re-ordering Rules:

The value of $P(C_2|(C_1, X))$ cannot be comprehended as the probability that chord C_2 follows chord C_1, since it is possible that C_2 cannot yield the required terminal. Rather, $P(C_2|(C_1, X))$ is the probability that the rule `"Any_X` → C_2`"` will be expanded before all other rules. If the chord that is chosen first does not succeed, the next chord is chosen via the same process. To give a better understanding, below is a simplified example.

Let us consider the following grammar:

$$G(V, \Sigma, R, S) = (\{A, B, C, D, X, S\}, \{0, 1\}, R, S)$$

Here, `A, B, C`, and `D` are "chords".

Let the rules of the grammar be the following:

S → AX

X → A | B | C | D

A → 0

B → 1

C → 1

D → 0

Let us define the weight matrix as follows:

	A	**B**	**C**	**D**
A	*40*	*10*	*10*	*40*
B	*50*	*10*	*10*	*20*
C	*10*	*10*	*40*	*40*
D	*50*	*10*	*10*	*20*

Table 5.2: *The weight matrix for the sample grammar.*

Let the input string be '01'. So, the derivation begins as follows:

$$S \Rightarrow AX \Rightarrow 0X$$

Since A is a "chord" that successfully yielded its corresponding terminal, we store it as the last chord encountered. The next step involves choosing one of the 4 rules of X to be expanded first. Since X can yield multiple "chords", we use the weight matrix to reorder its rules. We know that the previous "chord" is A. So, using the matrix at row A, the probabilities for (X→ A),(X→ B),(X→ C), and (X→ D) are $P(\text{A} \mid \text{A})$,$P(\text{B} \mid \text{A})$, $P(\text{C} \mid \text{A})$, and $P(\text{D} \mid \text{A})$ respectively. These values are $\frac{40}{100}$, $\frac{10}{100}$, $\frac{10}{100}$, and $\frac{40}{100}$.

Using these probabilities, the first rule expansion is chosen. If the chosen "chord" is B or C, the derivation is successful.

$$S \Rightarrow AX \Rightarrow 0X \Rightarrow 0B \Rightarrow 01$$

However, let us look at the case where A or D is chosen.

If X→ A is chosen first, it does not lead to a valid derivation, so we must choose another rule. Now the

probabilities of `X` yielding `B`, `C`, and `D` are 10/60, 10/60, and 40/60 respectively. This is because the remaining weights add up to 60.

To simulate this entire process in one run, the set of rules for `X` will be **re-ordered**. This entails choosing the first rule to expand, then choosing the next rule out of the remaining rules, and so on; recursively until the entire set has been re-ordered.

5.4.4 Scoring System for Derivations

Since the weight matrices convey the weightage associated with ordering certain rules, they only act as a table of probabilities. The resulting derivation is influenced by these weights because of the algorithms incorporated in the parse tree derivation. Using the weight matrices, it is also possible to assess to what extent the resulting derivation was consistent with the weight values. Consider the input, and the obtained accompaniment to be as follows:

Input: `happy 0 0 7 7 9 9 7 7 5 5 4 4 2 2 0 0`
Accompaniment: `0-M 9-m 7-M 4-m 2-m 9-m 0-M 0-M 5-M 5-M 4-m 9-m 2-m 7-M 0-M 0-M`.

In order to assess the accompaniment, we can look at every subsequent chord transition.
`0-M` $\rightarrow$ `9-m`
`9-m` $\rightarrow$ `7-M`
`7-M` $\rightarrow$ `4-m`
...
We then add the corresponding probability value, $P($`9-m|(0-M,happy)`$)$, etc. for each transition, and return the sum as the accompaniment's **score**. For every successful derivation, the score could be different. So, we can use the scores of multiple accompaniments to compare how well they characterize the given style.

5.5 Granroth-Wilding/Steedman Jazz Corpus

Section 5.4.2 describes the format in which our application stores transition weight matrices. A natural question that arises is the basis on which these transition weights were assigned.

The transition weight matrices corresponding to the **happy** and **sad** styles were constructed using our own basic knowledge of music theory, employing some of the concepts described in Section 2.4.

The matrix weights for the **jazz** styles, however, were constructed by performing data analysis on the

Granroth-Wilding/Steedman Annotated Jazz Chord Progression Corpus [1].

5.5.1 Dataset Description

This dataset consists of a total of **76 sequences** of chords of different jazz songs (all of which were transposed to the key of C), along with the beat duration of each chord, the original key signature, time signature, chord extensions etc, in CSV as well as human readable format. The corpus consisted of a total of over 3000 chords to work with.

5.5.2 Data Analysis

The primary analysis that we did with this dataset, was to track the frequency of transitions from one chord to another, for a particular beat position. We then identified the 20 most frequently occurring chords and used their stored transition frequencies to construct the weight matrices.

At the end of this analysis, we parsed a total of about 12000 chord transitions, and 101 unique chords and chord types.

6. Conclusions and Future Work

The link to the GitHub repository of the web application for this project can be found here:

GitHub Repository

The following points capture the vision for the project in the future.

(i) **More accurate transition weights backed by data:**
The transition weight matrices described are a crucial part of the application in terms of generating more accurate and *musically sound* chord accompaniments. While the current matrix weights are largely influenced by our own intuition and basic understanding of music theory, performing data analytics on the chord transitions of existing pieces of music can help improve these weights.

(ii) **Optimising the algorithm for parse tree generation:**
It is easy to see that the algorithm described in earlier chapters has an exponential time complexity, since we could potentially have to go through all possible branches of all variables before reaching a valid derivation. This is the reason our application is limited to generating accompaniment for melodies made of at most 16 notes.
A more efficient algorithm like the CYK Algorithm [3] can be employed to produce the parse tree in polynomial time.

(iii) **Support voice recording/ audio upload:**
For now, the only way a user can input a melody is by playing the melody using the inbuilt virtual piano interface/ manually selecting the notes in each bar of the melody.
A natural extension from this to improve user experience would be to introduce a way by which a user can record a vocal melody/ upload an audio track of a melody, which our application will then analyse and compute an accompaniment for.

(iv) **Other improvements to user interface:**
Expanding the range of keys in which the melody can be interpreted (for now, the application always assumes the melody to be in the key of C), adding more styles of accompaniment and exporting the application as an API are all incremental improvements that can make this project more versatile.

Bibliography

[1] MT Granoth-Wilding. *Harmonic Analysis of Music Using Combinatory Categorial Grammar*. URL: http://mark.granroth-wilding.co.uk/files/phd_thesis.pdf. (Doctoral thesis, accessed: 20.04.2021).

[2] N.Chomsky. *Syntactic Structures*. Mouton, 1957.

[3] Wikipedia. *CYK Algorithm*. URL: https://en.wikipedia.org/wiki/CYK_algorithm.