RiffShuffle: A Flexible Interface for Interactive Automatic Harmonization

by

Brian Chen

B.S., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author .....
Department of Electrical Engineering and Computer Science
May 12, 2020

Certified by .....
Eran Egozy
Professor of the Practice
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

RiffShuffle: A Flexible Interface for Interactive Automatic Harmonization

by

Brian Chen

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We design, implement, and evaluate a system that allows users to interactively generate a chord progression to accompany a given melody, with the philosophical goal of bringing the experience of composing to people with some musical experience. We use a simple Hidden Markov Model to drive the chord recommendation engine, but add multiple adjustable parameters to the model. We expose both established and novel interface elements that allow users to control these parameters and interact with the chord recommendations in other ways. We specifically investigate the interaction of randomly sampling chords from the Hidden Markov Model, and find that it is well-received by users. User studies suggest that users enjoy using the system, and that they are able to use it to create chord progressions efficiently in ways that reflect their own preferences.

Thesis Supervisor: Eran Egozy
Title: Professor of the Practice

Acknowledgments

I would like to thank the MIT music professors who agreed to speak to me during the early stages of this project and preventing me from being clueless about composition: Bill Cutter, Elena Ruehr, Matt Schumaker, Charles Shadle, Evan Ziporyn. Thanks also to all the instructors who forwarded my user study.

For all their support through this five-year journey at MIT, I would like to thank all of my friends, particularly Floorpi, ESP, and ✈✈✈ Galactic Trendsetters ✈✈✈. For this thesis I would specifically like to thank Mark, who offered valuable feedback early in the process. It is impossible for me to imagine my MIT life without all of you.

Special shout-out to Puzzlers Club for not only the stream of study participants, but also all the little interactions that kept me sane during quarantine. The middle-of-the-night electronic music party was crucial (5) for helping me finish writing this thesis in style.

I would like to express my thanks and love for my parents and sister for their eternal unwavering support and reminders to exercise.

And finally, my deepest gratitude goes to my advisor Eran Egozy, for agreeing to advise this thesis, suggesting the feature of the system that proved to be the most popular, and teaching the class that set me on this path three years ago.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Music composition has traditionally been an artistic process with a high barrier of entry. Experienced musicians may use years of knowledge in tonal harmony and counterpoint to compose novel melodies and chords, or they may improvise on an instrument to try out different musical ideas, requiring experience playing the instrument.

For computer scientists, particularly those working with artificial intelligence, this prompts the natural question: can computer systems be taught to compose music? Unlike many other fields of AI applications, which often have objective optimization goals, music composition is a creative and inherently subjective process. This has not deterred AI developers from creating many systems to generate music; indeed, it is a popular subfield of AI, as witnessed by the many high-profile recent systems for music composition and generation, such as OpenAI's MuseNet [15] and Jukebox [5], Google Magenta's Music Transformer [10], and Amazon's AWS DeepComposer [1]. These systems and many more have achieved impressive results in getting a computer to learn how to generate novel music, often only starting from a small human-suggested "seed" or nothing at all for each specific piece.

However, one may envision assigning the computer a different role in the composition process, and ask another question: can computers be taught to *assist* human composers in composing music? If one's goal is solely defined by the efficiency and quality of musical output, this question may not seem very different from the above

task, and could be solved with the same systems and designs. But if we consider composition as a creative activity that a novice human composer is participating in, pushing a button and having the computer spit out a full composition may not be so satisfying, because the human is not deeply participating in the process and lacks agency. This problem has received relatively less attention in the literature, and systems that do address it (such as MySong/SongSmith [16, 13], a major inspiration for this project that is discussed later in Section 3.1) often assume close to no formal music knowledge from the user.

However, even for would-be composers that have a little music training, there are many difficulties in composing a novel piece of music. In particular, chords and chord progressions are often more difficult to learn about and use adeptly in compositions. In Western equal temperament music, there are just twelve standard tones, which most music students learn about quickly; but these tones can combine into dozens of different chords, and there is no universally agreed-on list of all chords or ways to analyze them. Some chords, like the major and minor chords, are much more common than others, but there is a long tail of increasingly obscure possible chords to use — dominant, minor, major, and minor-major sevenths; diminished and augmented triads; augmented sixth chords of various nationalities — as well as ways to modify these chords — inversions, suspensions, the occasional sixth or ninth, and so on. In some sense, any set of notes played simultaneously can be considered a chord. Not only are chords harder to learn about formally, they are harder to identify when listening to a piece of music, and to play on most instruments. Most obviously, the human voice can only generate one note at a time (save for some quite difficult techniques like overtone singing). So, even novice composers who lack access to other physical instruments can often still experiment with melodies with relative ease, just by humming or singing; but they will have considerably more difficulty experimenting with different chords.

This difficulty, and the aforementioned perspective of composition as a creative process that we wish for human users to feel like active participants in, motivates us to create a system that can enable novice composers with some musical background

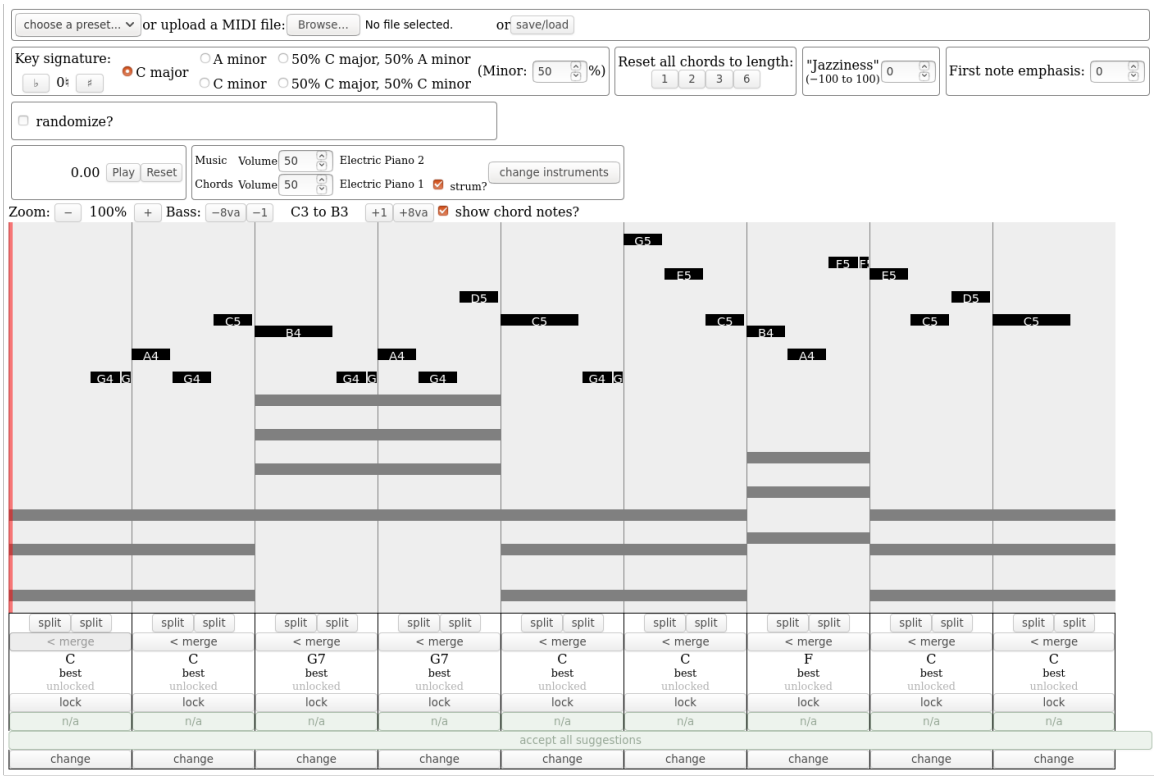

Figure 1-1: Screenshot of RiffShuffle harmonizing "Happy Birthday"

to compose chord progressions.

1.1 RiffShuffle

In this MEng project, we design, implement, and evaluate RiffShuffle, such a system for assisting users with limited musical experience in creating, experimenting with, and evaluating chord progressions to accompany a given melody.

RiffShuffle is a web application that allows users to upload a melody in the standard MIDI format, and then interactively suggests chords and chord progressions to accompany it. The interface, depicted in Figure 1-1, exposes several global parameters as settings that the user can adjust; after adjustment, new chord suggestions are dynamically recomputed based on the new parameters. The user can also manipulate individual chord suggestions to change them to other recommendations or an arbitrary supported chord from a longer list, which may influence the system to generate new suggestions. These two kinds of adjustments can be alternated, with

the user fine-tuning some chords and then changing global settings to explore other chords. The standard symbols are displayed for each note and chord so that users who understand them can interact and choose chords more deeply depending on their music theory background.

In addition, RiffShuffle has an option to randomly generate chord progressions instead of trying to suggest the optimal progression based on what it knows, which users can use as a source of inspiration to generate more unusual and personalized progressions.

1.2 Paper Overview

In Chapter 2, we discuss some of the background knowledge we use, including short explanations of music theory and the Hidden Markov Model from machine learning. In Chapter 3, we discuss prior work in computer-generated music similar to our goals by either generating chords or by intending to give users the experience of composition. In Chapter 4, we go over the design and implementation of RiffShuffle. In Chapter 5, we evaluate our system through user studies, with many examples of user-produced harmonizations. Finally, in Chapter 6, we discuss some takeaways and possible directions for future work.

1.3 Accessing the Implementation and Data

After publication, the code and data from the user studies, including the user-submitted harmonizations, will be available online at

`https://github.com/betaveros/riffshuffle`.

Chapter 2

Background

2.1 Notes and Chords

Music is an extremely broad subject, so just about any symbolic music system must set some kind of limitations or assumptions about the structure of music it can handle. Our system works with traditional Western music, in which notes in the melody and chords belong to one of the 12 standard pitch classes (C, C♯, etc,). In the following, we will try to discuss the system without assuming musical background from the reader when possible, but will occasionally assume that the reader is familiar with some basic music theory, such as note names, major and minor modes, and common intervals and chords. Readers who wish to brush up on music theory can consult any standard music theory textbook, such as [12].

Because the terminology and notation for chords is not completely standardized, some of the most common chord types and their constituent notes on top of the root note are summarized in Table 2.1. This table includes all chords our system is able to generate. Chords can also be inverted, and we will also encounter chords with unrelated additional notes in the bass, which are sometimes known as "slash chords".

Table 2.1: Summary of common chords and their names.

Chord	Notes		
major	major third	perfect fifth	
dominant seventh	major third	perfect fifth	minor seventh
major seventh	major third	perfect fifth	major seventh
minor	minor third	perfect fifth	
minor seventh	minor third	perfect fifth	minor seventh
diminished	minor third	diminished fifth	
diminished seventh	minor third	diminished fifth	diminished seventh
half-diminished seventh	minor third	diminished fifth	minor seventh
augmented	major third	augmented fifth	
power	(none)	perfect fifth	
pedal	(none)	(none)	
suspended fourth	perfect fourth	perfect fifth	
suspended second	major second	perfect fifth	

2.2 Hidden Markov Model

The Hidden Markov Model is a simple statistical model used in many fields of machine learning and pattern recognition. It is central to the design of RiffShuffle.

A *(discrete) Markov chain* is a randomized model consisting of a sequence of random variables $X_1, X_2, \ldots, X_n$. (As is common, we notate random variables with uppercase letters and values taken by those variables with lowercase letters.) This model generates a sequence of states $x_1, x_2, \ldots, x_n$, where each state x_{i+1} is chosen from a random distribution X_{i+1} depending on only the previous state x_i. We will focus on Markov chains that have a finite state space, so there are only finitely many possible values of each x_i, and that are *time-homogeneous* or *stationary*, so the distribution $P(x_{i+1} \mid x_i)$ does not depend on i. (We will also focus on discrete Markov chains; it is also possible to consider Markov processes where the states are a continuous function of time rather than a sequence.)

In the standard *hidden Markov model* (HMM) problem, we have a Markov chain (X_i) whose values cannot be directly observed (so its values are "hidden"), and a sequence of *observations* that are random variables (Y_i), where each Y_i is a random distribution that depends only on X_i. We also assume this dependence of Y_i on X_i

does not depend on i. Each of these probability distributions may or may not be known. For fixed x_i, the probability of X_{i+1} taking on a certain value is called a *transition probability* and the probability of Y_i taking on a certain value is called an *emission probability*.

We will primarily work with a small extension of the standard hidden Markov model where there can be multiple observations $Y_{i,j}$ corresponding to a particular state X_i in the Markov chain, but for a fixed i all variables $Y_{i,j}$ are independent, and again the distribution from which each $Y_{i,j}$ is drawn depends on neither i nor j.

2.2.1 Training

If many examples of sequences of states (x_i) and corresponding observations $y_{i,j}$ are known, the maximum likelihood estimates for the transition probabilities and emission probabilities for the Markov model can be calculated in the most obvious way, simply as the fraction of relevant transitions or observations that match.

That is, if the number of times state a transitions to state b is $t_{a,b}$, then the transition probability $\tau_{a,b}$ of some state a to some state b is just estimated as

$$\tau_{a,b}^{\mathrm{ML}} = \frac{t_{a,b}}{\sum_c t_{a,c}}, \tag{2.1}$$

the number of times a transitions to b divided by the number of times a transitions to any state. Similarly, if the number of times observation y is made during state a is $m_{a,y}$, then the emission probability of some observation y given some state a is estimated as just

$$\mu_{a,y}^{\mathrm{ML}} = \frac{m_{a,y}}{\sum_z m_{a,z}}, \tag{2.2}$$

the number of times y is observed during state a divided by the number of times anything is observed during a. This is because the independence assumptions of the Markov model reduce each parameter to simply estimating the probability distribution for a variable that can take on one of several discrete values, given many observations from that variable. See e.g. equation 2.33 of [4] for a rigorous derivation

of this maximum likelihood estimate.[1]

Finally, a Markov model also requires a probability distribution for the very first state X_1. We assume the distribution is stationary and estimate this distribution analogously to all the other probabilities above, by setting the probability of any state x being the first to be the fraction of occurrences of x across all sequences.

2.2.2 The Viterbi Algorithm

Suppose all transition probabilities and emission probabilities, as well as the initial state probability distribution, are known or have been estimated. Also suppose we are given a sequence of observations

$$y_{1,1}, y_{1,2}, \ldots, y_{1,k_1}, y_{2_1}, \ldots, y_{2,k_2}, \ldots, y_{n,1}, \ldots, y_{n,k_n}$$

along with how they correspond to a sequence of n states, in which k_i observations correspond to the ith state. Then the standard *Viterbi algorithm* (see e.g. Section 13.2.5 of [4]) can be used to find the most likely sequence of hidden states $x_1, \ldots, x_n$ that would have resulted in those observations. The basic idea is to use dynamic programming to compute, for each i and each possible value of x_i, the sequence of states $(x_1, \ldots, x_i)$ leading up to that value with the maximum likelihood, as well as that likelihood itself.

The Viterbi algorithm is reasonably efficient, but its runtime is quadratic in the number of states. Assuming all transition and emission probabilities have been precomputed and can be looked up in constant time, if there are s possible states, the Viterbi algorithm's runtime is $\Theta(ns^2 + s\sum k_i)$.

[1]Note that this problem is substantially simpler than what is more often referred to as the hidden Markov model problem in the machine learning literature, in which the sequence of states (x_i) is *never* observed, not even in the training data. That problem is discussed in Section 13.2 of [4]; no efficient exact solutions are known for it.

2.2.3 Application to Automatic Harmonization

The hidden Markov model can be applied to automatic harmonization by modeling the chord progression as a Markov process, i.e. each chord is drawn from a probability distribution that depends only on the chord before it, and the notes in the melody during each chord as observations, i.e. they are drawn from a probability distribution that also depends only on the chord.

Given a corpus with many examples of melodies and chords, we can therefore create a hidden Markov model simply by counting up the number of occurrences of each chord, transitions between each pair of chords, and number of times each melody note appears during each chord. Then we can compute the model's probabilities with equations (2.1) and (2.2). After this precomputation, if we are given a melody as well as the time at which each chord transition occurs, the problem of finding the most likely chord progression reduces to the standard problem of finding the most likely sequence of hidden states. This can be solved quickly and exactly with the Viterbi algorithm, as mentioned above.

Chapter 3

Prior Work

The problem we are considering, of designing a system to help users find chords to harmonize a given melody, can be classified under the field of *musical metacreation* [14], which concerns programming machines to have the ability to accomplish musical tasks. The field is large because of the variety of musical tasks, including composition, improvisation, accompaniment, and many others, but it is united by the difficulty of computerizing musical intuition, which is essential to musicians performing these tasks but is not at all easy to formalize. Because of the size of the field, we will focus on works related to either our explicit technical goal of generating accompaniment from melodies, or our philosophical goal of helping non-composers or inexperienced composers experience composing music.

3.1 MySong/SongSmith (Simon et al.)

One of the projects that matches both criteria in this field, which is also a major source of inspiration for RiffShuffle, is MySong [16, 13], a tool from Microsoft Research that generates accompaniment for vocal melodies. At its core, MySong uses the relatively simple Hidden Markov Model, discussed in Section 2.2, to output the most likely sequence of chords for a given melody. The model was trained on 300 lead sheets following essentially the procedure in Subsection 2.2.1.

MySong's primary innovations were not in improving the sophistication of its ma-

chine learning model per se, but rather, in providing a user-friendly interface on top of the model, which allows the user to adjust parameters of the model and other aspects of accompaniment. Furthermore, this interface was designed to be intuitive even to users with no formal knowledge of machine learning or music composition. Specifically, MySong exposed two sliders described as controlling the "happy factor" and "jazz factor", respectively, which changed weights inside the chord recommendation algorithm. These two settings were incorporated roughly as-is into RiffShuffle's model, and are discussed in Subsection 4.3.1. However, the internal workings and technical meanings of these factors were not exposed to the user. Through empirical studies, the designers of MySong found that users were nevertheless able to quickly get an intuitive sense of those two sliders. When asked the free-response question "What things were most useful about this system?" many users specifically mentioned these sliders, calling them "useful for learning" and "easy to understand" [13, p. 7].

Another aspect of MySong designed to minimize the barrier to entry is that it receives the melody through vocal input, which means users can provide a melody to the system with no knowledge of musical notation or software at all. The user's singing is pitch-tracked to produce a histogram of the 12 fundamental pitch classes that is fed into the HMM as input. This input method greatly improves the accessibility of MySong to inexperienced users. However, the design choice comes at the cost of some precision for more experienced musicians, who may wish to try harmonizing a more complex melody they cannot replicate with their voice easily.

One final interaction supported by MySong is that it allows users to adjust and "lock" specific chords to chords other than the system-recommended ones, thus giving users much more fine-grained control over the chord progression, but this feature is best for somewhat more musically experienced users who understand the names of chords.

After its initial publication, MySong was further developed and then renamed SongSmith. SongSmith can still be downloaded for a free trial online from `http://songsmith.ms/`, although it only runs on Windows.

3.2 Improved Models for Chord Accompaniment

Since the publication of MySong, many researchers have designed improved machine learning models for recommending chord progressions given a melody. Sometimes, these models are evaluated objectively with standard accuracy metrics from machine learning; in other cases, they are evaluated subjectively by human judges listening to the results. These research projects are relevant to the machine learning aspect of RiffShuffle's design.

3.2.1 Hidden Semi-Markov Model (Groves)

Most directly, Groves [6] points out limitations of MySong and its model, and extends it to avoid them. The limitations include that MySong ignores the individual durations, onset timings, and order of the notes in the melody, and that it fixes the length of every chord to one full measure. To overcome these weaknesses, Groves develops an alternative model using the more flexible Hidden Semi-Markov Model (HSMM), and evaluates it on Temperley and deClercq's Rock Corpus [3], a database of 200 rock song melodies and harmonizations that we will see repeatedly. The difference between the HSMM and the HMM is that in an HMM, the probability distribution of the next state can only depend on the previous state, whereas in an HSMM, the probability distribution of the next state can depend on the previous state as well as how long that state has been occupied by the HSMM. When applied to chord recommendation, the result is that the HSMM can also learn and recommend where chord transitions should go, whereas the HMM depends on the timing of every chord transition being part of the input. Groves primarily evaluates his model with standard objective data-based metrics in machine learning, cross-validating with partitions of the test data.

3.2.2 Long Short-Term Memory Neural Networks (Lim et al.)

Lim et al. [9] propose a more radically different model, using neural nets with the bidirectional long-short-term-memory (BLSTM) architecture to generate chords from

a melody. BLSTM neural networks are a type of recurrent neural network, meaning they feed back into themselves and can handle inputs and outputs of variable sizes. The long-short-term-memory architecture is specifically designed to capture long-term dependencies in the network's inputs and outputs in a controlled, practical manner, and the bidirectionality means the dependencies can go forwards or backwards. This makes sense in a musical context, since one can imagine motifs or chord progressions in a piece of music that only repeat in very faraway locations. The model is trained on a database of 2252 lead sheets they gathered from the now-defunct website `wikifonia.org`.

The authors test their BLSTM recommender against a simple HMM model as well as a deep neural network–HMM (DNN-HMM) hybrid model popular in speech recognition, using both an objective data-based method and a subjective test with human judges. The objective comparison shows that the BLSTM is more accurate than the other models. Empirically, through a "confusion matrix", they observe that the HMM's outputs are heavily skewed towards the common I, IV, and V chords, at the expense of other chords. Through user evaluation, they find that the BLSTM provides significantly more highly-rated results than the HMM or DNN-HMM model, and that the gap is wider when users were familiar with the melody. However, the user ratings for the BLSTM's output are also still significantly less than the "ground truth" harmonizations taken from the data set.

3.2.3 Comparative Study (Yeh et al.)

One of the most recent studies for automatic chord recommendation (published during the preparation of this thesis) is a comparative study by Yeh et al. that compares and builds on several approaches, including the HMM and BLSTM discussed above but also a conservative template-based algorithm and a genetic algorithm. They analyze all of these established algorithms and, after identifying some weaknesses in the BLSTM, develop an additional novel approach they call the MTHarmonizer, which extends the BLSTM model to also predict chord function (tonic, dominant, or subdominant chords) as an intermediate step.

Another major aspect of progress is their much larger training dataset, which they call the Hooktheory Pianoroll Triad Dataset (HTPD3). It is the result of standardizing a collection of 11,329 lead sheets, and represents a more than 5× increase in training data size over the above efforts (although, as a point of comparison, OpenAI's most recent Jukebox [5] used a novel dataset of 1.2 million songs in raw audio format).

The authors compare their different models on both objective and subjective metrics. Objective evaluation was done both through chord accuracy as well as through several different metrics quantifying the relationship between melody and chord. Subjective evaluation was performed through an online survey with 202 participants ranking several harmonizations of the same melody generated by different approaches. They found that out of the approaches they tested, the MTHarmonizer achieved the best accuracy and the highest subjective scores. However, they note that the genetic algorithm was evaluated unusually well along the "interestingness" metric, and suggest that a hybrid genetic algorithm with the MTHarmonizer could be worthy of future research.

3.3 Tree-Structured Generative Model (Tsushima et al.)

Returning to chord-recommendation models with novel user interfaces, Tsushima et al. [17] take a somewhat different approach. They propose a tree-structured model that generates chords based on a probabilistic context-free grammar (PCFG). Users can interact with the tree by splitting and merging nodes, which prompts the machine learning model to regenerate chords by applying the PCFG following the new tree. This is an interesting different direction in which users can provide the machine learning algorithm with input about the high-level structure of the chord progression, rather than about individual chords or properties of individual chords. They evaluate the model and interface through objective measures and user tests, and find that users generally rate each of the interface's operations to be useful. They also find that even

users with no musical background are able to interact with the tree-based model.

3.4 ChordRipple (Huang et al.)

A paper with a goal that is technically quite different, but that is spiritually related to our goal of letting users experience creative composition, is Huang et al.'s ChordRipple [7]. This is a pure chord recommendation system that does not work with a melody at all. Given an input chord progression, ChordRipple generates and suggests modifications to the chord progression, with a focus on creativity and "unusual" suggestions. Its interface enables users to interact with and adapt the different variations freely, which provides users with agency in composition. Huang et al. also run a variety of user studies to evaluate ChordRipple, one in which freeform feedback is solicited from users and one in which users tried different variations of the ChordRipple system and the "novelty" of the generated chord progressions was quantified and compared. They conclude that their system does have an effect in causing users to choose more adventurous chord progressions. Thus, their paper provides a good example of a mechanical system fostering creativity.

3.5 The Bach Doodle (Huang et al.)

One of the interactive musical metacreation tools that may have reached the widest audience is Google's Bach Doodle [8], which was released on March 21, 2019, on Bach's 334th birthday. It is an interactive toy that receives a soprano melodic line as input and generates three other parts to form a four-part harmony. The system used 306 Bach chorales as its training corpus; although this dataset is relatively small, it has a much more uniform and tightly-regulated structure, which is likely to make machine learning easier.

Although the Bach Doodle is designed for a wide audience, it actually requires more musical experience to fully use and grasp than most of the other tools we have examined, since the user inputs a melodic line by clicking notes to place them on

sheet music. However, the interface is simple enough that users can enter *some* melody without knowing music notation, and can influence the overall shape of the melody. In addition, the tool provides three familiar preset melodies that users can enter with a single click and without any musical knowledge, so that those users can still get to interact with the tool. Although there are few interactions the user can perform beyond inputting the melody, the user can change the melody and ask the system to reharmonize it with relatively little effort. They can also ask the system to reharmonize the same melody, which produces a new harmonization because as the system is randomized.

The authors evaluate the Doodle through soliciting user feedback for each produced harmonization. The Doodle asks users for a positive, neutral, or negative rating after each harmonization. When the study was published, a majority of all harmonizations were positively rated, including even harmonizations that users did not submit a rating for. This suggests that users liked the harmonizations. However, they also identify one specific weakness based on chorale rules and freeform feedback from advanced users, which is that the model generated parallel fifths and octaves (P5s and P8s) fairly often, even though they are generally forbidden in Bach's counterpoint and only sometimes excused in specific circumstances. The authors suggest that their training data was unfortunately imprecise enough to capture those circumstances, so the model learned to generate P5s and P8s more permissively than Bach would.

3.6 Other Projects

As mentioned, these are just some of the most technically and spiritually related works in the area; for a more comprehensive review of the literature, we refer the reader to Makris et al. [11] or Pasquier et al. [14].

Chapter 4

Design and Implementation

4.1 Philosophy and Goals

As we have seen in the previous section, various systems for automatic harmonization have been extensively studied and evaluated, often achieving high accuracy and generating progressions that receive high subjective ratings from human judges. Relatively less attention has been paid to the interactive usage of such tools for composition, or to the subjective experience of composers when using such tools. The studies and research efforts that do focus on interactive usage have usually focused on users with minimal or no musical experience. However, we believe there is a substantial skill range of amateur musicians who know enough music theory that they can interact more deeply with a musical model, but could still benefit from some kind of computer assistance, for either creativity or efficiency in composition.

Therefore, RiffShuffle focuses on innovating in the human interaction and composition part of the process and differentiates ourselves by targeting users with slightly more musical experience. Thus, we generally assume that users are comfortable with names of musical notes, chords, and keys, e.g. C major and C minor, and freely expose interface components and labels with those names.

In preparation, we also interviewed several experienced composers at MIT to learn more about their composition process. It was only possible to incorporate a small fraction of the suggestions and feedback we received into the present project, and

some directions for potential future work suggested by the results are described in Chapter 6. However, generally the interviewed composers agreed that a large part of the reward of composition is the novelty and creative control that the composer can exert over the composed piece. Since our system is designed in no small part to help users with limited music background experience the joy of composing, it is important that the developed system be flexible and give the composer this form of creative control, instead of simply rigidly imposing rules it learned from existing music.

4.2 Data

As with all machine learning systems, our system needs to derive its musical rules from somewhere. For efficiency of implementation, we use existing corpora of chords and melodies, and train our system on them. We synthesized several sources referenced by previous papers to serve as our training data.

The first is a dataset of 200 transcribed rock songs with melody and chords compiled by Temperley and deClercq. The regularity and high quality of the dataset has made it a popular target for music research; for example, Groves [6] and Tsushima et al. [17] both also derive training data from it. Originally, the dataset comprised 100 rock songs selected from "500 Greatest Songs of All Time", a list published by Rolling Stones, and was published as "A corpus analysis of rock harmony" [3]. The corpus was later expanded to 200 rock songs and is freely available online in several formats at `http://rockcorpus.midside.com/`. Although the list is labeled as rock songs, it has more musical diversity than one might expect. The authors acknowledge they were "puzzled" by some of the songs included on the list, including songs that might usually be classified as jazz, rap, country, and other genres. However, this is ultimately a reflection of the fact that rock is a loosely-defined genre of music. For each song, Temperley and deClercq's data set contains the Roman numeral analysis and onset of each chord, as well as the pitch and onset of each melody note. Note that there are two chord analyses of each song, one for each author, but only one melody transcription, by one of the authors depending on which song. To produce training

data that is internally consistent, we only use the chord analysis performed by the same author as the melody transcription on each song. Following the authors' description of their analysis, we write a manual parser to convert each Roman numeral into a set of pitch classes.

The second dataset, which has more songs and no genre restrictions, is a collection of lead sheets collected from `wikifonia.org` by Lim et al. [9] to train their machine learning model. It is available at `http://marg.snu.ac.kr/chord_generation/`. This data set is much larger, with 2252 lead sheets, and comes in a tab-separated format where each measure, chord, and melody note is labeled. However, it only includes pieces in a major key and unfortunately has a few quality problems, which are briefly discussed in Subsection 4.2.1.

Finally, we use the Nottingham Music Database, a collection of 1034 folk songs that has had multiple maintainers. This dataset was also used by Tsushima et al. [17] for their tree-structured chord sequence interface. Compared to the other data sets, the chords in this data set are relatively simple, with the caveat that slash chords (chords with a different bass note specified via appending a slash and another note) appear commonly, and occasionally the bass note is also a fairly unusual non-chord tone (examples include "D7/b" and "D/g"). We parse these from the original format into notes and chord symbols using the open-source music library `music21` [2].

The benefit of a diversity of sources of training data is that it gives the model more flexibility, and prevents the model from learning and imposing an overly rigid style. In addition, one initially considered idea for an interactive feature was to expose the different data sets and allow users to choose or weigh which data sets would be incorporated into the model's suggestions, but this was not implemented to keep interface complexity limited.

4.2.1 Analysis

One fairly important feature of every song is its key and mode. When training our model, it is important to take these features into account, because they greatly influence which chords are more likely to appear; as an extreme example, F♯ chords

are unlikely in a C major song, but very reasonable in a F♯ major song. However, the key and mode information is not always available in the standard music notations we encountered and occasionally must be derived. To train MySong, for example, Simon et al. created a simple predictor to classify each of their lead sheets as major and minor. Fortunately, the key and mode is freely available in the Nottingham Music Database's `abc` format, but some effort and judgment was required to determine the key and mode of the other songs in our other datasets.

Somewhat perplexingly, although Lim et al. write that they collected 2252 lead sheets "all in major key", their dataset, which is in a simple tab-separated text file format, has a `key_mode` column that occasionally has the label `minor`. This does not have a large effect, however, because only 45 songs have a "minor label", which is a small fraction. In addition, some of the songs likely have mislabeled modes, which we discovered while looking at the songs with the highest frequency of certain chords. Based on some manual investigation, the most likely explanation is that some scores lacked a key signature and were labeled as C major, when the melody and chords were really in a different key. For example, the songs "Ievan Polkka" and "When Johnny Comes Marching Home" were both notated in C major, even though the notes were in D minor. "Tell It On the Mountain" and "Oh My Darling, Clementine" are notated in C major while the chords and melody are in F major. However, because these issues seemed to be minor and it was impractical to go through all the songs to find mode problems, we used the dataset as-is.

The biggest challenge for this determination is in the rock song corpus, in which Temperley and de Clercq made the deliberate choice to notate only the tonal center of each song, and not the mode. They write, "It is often difficult to categorise rock songs in the common-practice tonal systems of 'major' or 'minor'; many songs adhere to other modes (such as Mixolydian), freely mix triads from the major and minor modes, or even combine elements of major and minor simultaneously (e.g. a minor third in the melody over a major triad)."

Although we agree that modes may be ambiguous in theory, we find that with some methodology similar to that used in the paper, we can crudely classify most

songs as major or minor without significant issues. One analysis Temperley and deClercq run is to build "chord vectors" by checking the presence and absence of each chord root (of 12 choices, relative to the tonic), and look for correlations between the presences of different chord roots. If we instead classify all chords as major or minor in additional to by chord root, so that there are 24 distinct classifications, we find that by far the strongest correlation is negative between the I and i chords. This suggests that although not every song can be neatly classified as major and minor, there is still a strong qualitative difference between songs in major and minor key, and this is one of the sharpest ways to qualitatively classify the rock songs into two categories. Indeed, of the 200 songs, only 13 had both I and i chords (including chords built on top of those triads, such as with sevenths or suspensions). A list of the strongest correlations is provided in Table A.1. This observation motivates us to simply classify songs as minor if they have a minor i chord.

4.2.2 Data Preprocessing

After ensuring our training data includes modes for all songs, we still need to make several decisions to convert songs into a format suitable for training.

4.2.3 Chord Simplification

If we try to use the chords in our datasets directly, we find that the number of different chords is too large. This makes the HMM slow, due to the quadratic factor of the number of chords in the Viterbi algorithm's runtime, as seen in Subsection 2.2.2. It also hinders learning by reducing the number of data points the model can see for each chord. Thus, every paper on the subject we have found does some simplification and canonicalization of chords and melody before training their model on the resulting data. Several simplifications are just about universal, including transposing all pieces of music to the same key and removing all inversions, but there is no strong consensus on exactly how to simplify the chords further:

- MySong [16] simplifies each chord to its core triad (major, minor, augmented,

diminished, and suspended) and uses "start of song" and "end of song" as two "virtual" chord types, resulting in $5 \times 12 + 2 = 62$ chord types. The "suspended" category of triads consists of "suspended-seconds, suspended-fourths, and chords appearing with no third" (sometimes known as a "power chord"); unfortunately there is no clear description of what notes the system actually played for such a chord.

- Groves [6] similarly removes all inversions and additions to triads except for dominant 7th chords, which "have implications of the key tonality, and thus could be considered to be fundamentally different from a simple tonic triad." However, note that Groves works directly with Roman numeral analyses and not with note triples, meaning chords consisting of the same notes that can be analyzed as different Roman numerals (e.g. V7/IV and I7) could be analyzed differently.

- Tsushima et al. [17] go the furthest of papers we found, only considering major and minor chords, for a total of 24 chord types.

- Yeh et al. [18] also reduce chords to triads, including "major, minor, diminished, and augmented chords without 7ths or additional extensions," while also including a "no chord" chord (N.C.), giving 49 chords. Suspended chords are handled by "mapp[ing] to the major and minor chords"; it is not entirely clear what this means.

In the first iteration of RiffShuffle, we followed Groves's approach the most closely, reducing all triads to major and minor chords and eliminating all inversions and additions except for the dominant 7th, because we agree with his harmonic reasoning the most. In addition, we have an empirical reason, which is that in all three of our corpora, major, minor, and dominant 7th chords are the three most common chord qualities in some order, and this remains true after some types of inversions and additions are eliminated. This does not necessarily mean that those chord qualities are the most harmonically distinctive, but it does mean that our HMM has more

examples of dominant 7th chords to learn from than it would for, say, augmented chords if those chords were separated out. This can reduce overfitting. In addition, we kept the N.C. and pedal chords due to their rarity and the lack of any clear chord to map them to. Augmented and diminished chords were treated as major and minor chords, respectively; also, for simplicity, power and suspended chords were just mapped to major chords.

The frequencies of chord qualities from our three corpora, which are the basis for our empirical argument, are tabulated in Tables A.3 to A.4. (As several of the chord quality notations were ambiguous or poorly formatted, several subjective calls were made in normalizing the chord qualities to the ones presented, but the observed differences are stark enough that they seem unlikely to be affected by these subjective calls.)

In the second iteration, after user feedback specifically mentioned that users wanted to use minor 7th chords and diminished chords, we expanded the set of output chords to also include those two types. All other chords were still mapped as before.

4.2.4 Transposition

Another ambiguity we must resolve is what it means for all songs are transposed into the same key.

MySong transpose lead sheets all into the same "key", i.e. key signature, of 12 possibilities (set of sharps and flats, up to enharmonic equivalence, or equivalently, the set of notes in each scale). This groups songs in a certain major key with songs in that key's *relative minor*, e.g. C major with A minor. However, under the Temperley and deClercq paradigm, keys are defined by their tonic, so that songs in a major key are categorized with songs in its *parallel minor* key instead, e.g. C major with C minor. Both choices could be considered advantageous for some songs and styles of composition. Tsushima et al. [17] and Yeh et al. [18] actually also transpose all pieces of music to C major and C minor.

We struck a middle ground here, leaning towards the MySong idea of key signature, as this seemed empirically closer to what we could determine, but providing both as

options. This is strongly influenced by the standard MIDI format that we take as input, because the MIDI format has a field for key signature (number of flats or sharps) and a field for mode (major or minor), but the latter is less popularly generated by tools that could be used to create a MIDI file. For example, Musescore, a popular open-source software scorewriter that was also used to generate many test melodies, allows users to generate scores and MIDI files with any key signature from 7 flats to 7 sharps through its interface, but does not allow users to specify the MIDI file's mode (major or minor). Thus, it is more likely that a user would want to switch from a major key to its relative minor rather than its parallel minor.

4.3 Model

After normalizing the musical input, we use it to train the Hidden Markov Model described in Section 2.2 and also used in MySong/SongSmith. Although we have discussed several improved models in Chapter 3 that would likely produce higher quality chord recommendations, RiffShuffle still uses the simple HMM, chiefly for performance reasons. Firstly, compared to other models, it is extremely fast to use, enabling user interaction at a much faster rate. A good interactive tool should respond quickly to user input, and more heavyweight ML models might not be able to recalculate outputs quickly enough to user input or changing parameters, or at least require additional performance engineering to achieve an acceptable speed. One can see an example of such performance engineering in Huang et al.'s Bach Doodle [8], in which the initial browser implementation required over 40 seconds to harmonize each melody, and the authors only brought it down to 5 seconds with considerable performance engineering and work with WebGL. With the simple HMM, we are less constrained by performance and have more freedom to consider and implement different interactions.

As discussed in Subsection 2.2.1, the HMM is also very easy to "train". All we need to do is count how often each chord transitions to each other chord and how often each melody note appears during each chord. This also makes it easy to interpolate

between models trained on different data sets, although the result of interpolation may not necessarily have any mathematical meaning. Another advantage is simply that it is easier to adapt existing interface elements in MySong/SongSmith to our project so we can evaluate their usefulness with a different user audience. Future work could explore exposing interaction of more advanced models, including models that have already evaluated in other ways.

Another advantage of the HMM specific to our interactive paradigm is the localized effects of individual chords. That is, if we change one chord but keep the melody and all other chords fixed, it only affects a few terms in the probability that the HMM assigns to the chord sequence and melody: the probability of the observations during that chord, and the probability of the previous and next chord transition.

For a more concrete example consequence, suppose the HMM finds the most likely chord sequence (x_i) for a given melody, and we then ask it to find the most likely chord sequence but with the constraint that for some specific j, the jth chord is fixed to some other value x'_j. If the HMM finds the chord sequence (x'_i), and it matches the original chord sequence $x_k = x'_k$ at some location $k > j$, then $x_\ell = x'_\ell$ for all $\ell > k$. Similarly, if $x_k = x'_k$ at some $k < j$, then $x_\ell = x'_\ell$ for all $\ell < k$.

One intuitive way to describe this would be that this prevents "action at a distance" in the model, in which chords affect other chords that are very far away. In a noninteractive paradigm, action at a distance is probably actually desirable for finding the overall optimal chord, since many songs have long-term structure that good models should capture. Indeed, LSTM neural net architectures are designed precisely to allow inputs to have manageable effects on distant outputs. However, such effects can be disorienting to users who do not expect interacting with the model at one place to change all of the model's outputs.

4.3.1 Adjustable Model Parameters

As mentioned above, two of the adjustable parameters of our model correspond closely to factors in MySong. The "happy factor" causes interpolation between major and minor chord recommendations, and is implemented in much the same way. We follow

MySong in using linear interpolation in the log domain, which does not exactly correspond to any neat mathematical identity, but appears to work empirically better. That is, if the estimated transition probabilities from chord a to chord b are $t_{a,b}^{\text{maj}}$ and $t_{a,b}^{\text{min}}$ in the major and minor datasets, respectively, and the happy factor is $h \in [0, 1]$, then the model uses the transition probability $t_{a,b}$ that satisfies

$$\log t_{a,b} = h \log t_{a,b}^{\text{maj}} + (1 - h) \log t_{a,b}^{\text{min}}.$$

The same interpolation is performed for emission probabilities. However, we will point out a potential disadvantage, which is that impossible events in one data set are punished infinitely harshly compared to other events, which would not occur with linear mixing. Since we expect our target audience to understand major and minor keys, the parameter is directly described in the interface as controlling to what extent the mode is major or minor. Finally, we also include two variants that the user can choose from, one that interpolates between a major key and its relative minor, and one that interpolates between a major key and its parallel minor, due to the ambiguity described in Subsection 4.2.4.

The "jazz factor" causes a changing weight between melody and chords. That is, given a number $j \in [-1, 1]$, when the log likelihood for some sequence of chords is computed as a sum of log transition probabilities and log emission probabilities, all log transition probabilities are multiplied by $1 - j$ and all log emission probabilities are multiplied by $1 + j$. This parameter is called "jazziness" and explained in a tooltip because, although it is based on concepts that should be familiar to users, it is not a single unified concept that users will be familiar with.

One new parameter we added was a "first note emphasis" parameter $f \in [1, \infty)$ in order to allow our model to take into account note order. The simplest way to explain this parameter is that it is equivalent to making the model pretend that the first note in every melody is played f times instead of just once, although f can be fractional. That is, when training, the first note in every measure counts as f observations instead of 1 for the purpose of estimating emission probabilities (fortunately for our model's

efficiency, pre-summarization is still possible because we can simply count the number of note appearances that are and are not the first note in each measure separately). Later, when running the Viterbi algorithm, the log emission probability for the first observation in every measure is multiplied by f when computing the likelihood of any chord progression. Although this parameter is useful in some cases, it also often behaves counterproductively when the first note is a suspension from the previous chord. One avenue of exploration here that would require more work is to develop an interface that allows users to experiment with varying the weight of each individual note in a measure or to assign them to different measures.

Another parameter that was under consideration but ultimately not implemented was one that controlled the importance of note duration: at one extreme, all notes would be weighted equally regardless of duration, and at the other extreme notes would be treated as a fractional number of occurrences based on their duration. This was not implemented for two reasons. One is that note duration was not available in Temperley and deClercq's Rock Corpus, which only tracked the onsets of each note, so we would need to decide how to approximate durations for those notes in order for our model to train on that data. The other reason is that it is harder to pre-summarize the training data in a way that allows us to dynamically and efficiently regenerate an interpolated model based on the changing parameter.

That is, if we had a "duration factor" $d \in [0, 1]$ that controls how important duration of melody notes is, we would probably want a note of length x to count as 1 occurrence if $d = 0$ and x occurrences if $d = 1$, for both training and chord generation. The natural way to interpolate would thus be to make the note count as x^d occurrences. However, there is no way to efficiently summarize a list of x's that allows us to dynamically calculate $\sum_x x^d$ for any d except quite inefficiently by storing the multiset of all x's, and we need this sum to generate the model. A natural workaround would be to interpolate linearly, i.e. make a note of duration x count as $1 - d + dx$ occurrences; however, this is a much less coherent interpolation strategy because 1 is unitless but x is a duration. Because of these concerns, we did not implement such a duration parameter to our model, but future work could

investigate whether it is possible.

4.3.2 Randomization

A more significant novel feature of the model is its support for randomization. In this mode of operation, the model no longer tries to compute the optimal (maximum likelihood) chord sequence, but simply samples a random chord sequence by choosing a chord randomly at each step according to the computed probability distribution. The model also exposes a random seed as an argument that can be passed in, so that the random process can nevertheless be made deterministic.

We also expose a new adjustable "determinism/chaos" parameter that only affects the randomization process. This parameter takes on a value $r \in [0, \infty)$ that is by default 1 and is used as follows: Whenever we randomly choose a chord, we compute the probability of every possible chord, exponentiate each of them by the parameter r, renormalize the results by multiplying by a constant so they sum to 1, and treat these normalized numbers as the actual probabilities of the distribution we sample from. Similar to the "happy factor", we are not aware of any neat mathematical interpretation of this exponentiation process, but believe that it makes intuitive sense. Its effect is that lower values of r make the model relatively more likely to choose chords that were considered less likely. At the extreme when $r = 0$, all chords the model knows about are equally likely regardless of chord progression and melody, so the model is simply outputting a "progression" of completely random chords; intuitively, this is "maximum chaos". When $r \to \infty$, the most likely next chord is always chosen, making the process completely deterministic and minimizing "chaos". Note that this latter setting is not equivalent to the maximum likelihood result found by Viterbi's algorithm, as it is the analogue of the greedy algorithm while Viterbi's algorithm performs dynamic programming.

4.3.3 Chord Progressions For Runner-Up Recommendations

Finally, our model also allows "chord-locking", that is, it allows certain chords at certain indexes to be fixed. This is easily performed in the Viterbi algorithm simply by considering the locked chord to be the only possible candidate when computing probabilities for the chord at the index after it. However, one less obvious consequence of the HMM's simplicity is that the Viterbi algorithm also allows us to effectively compute, for every index i, the maximum likelihood sequence of chords where the chord at that index were instead fixed to some other specific chord. This can be done for all indexes and all alternate chords while only requiring a constant factor increase in runtime.

To go into this more deeply, we must also explain Viterbi's algorithm in more detail. The standard formulation of Viterbi's algorithm as applied to an HMM calculates the sequence of states $(c_i)_{i=1}^n$ that maximizes the likelihood of that full state sequence $P((c_i)_{i=1}^n)$ and observations matching what is known. It does so by performing dynamic programming and computing, for each index j and state c', the sequence of states $(c_i)_{i=1}^j$ with the maximum likelihood where $c_j = c'$, as well as that probability, which we might call $P_{j,c'}^{\rightarrow}$. This partial maximum likelihood takes into account all transitions and observations up to and including chord j; in our setting, where there can be many observations per chord, it also makes sense to precompute e_{j,c^*}, the likelihood of the observations during the jth chord actually occurring during chord c^*, for all j and c^*. Also, we do not actually need to store the full sequence $(c_i)_{i=1}^j$ for each pair (j, c'); simply storing the value of c_{j-1} that produced this maximum likelihood will suffice, as we can follow those stored values backwards from any c_j to reconstruct the full sequence.

Viterbi's algorithm can be run backwards equally well, in which case the dynamic programming computes, for each index j and state c_j, the sequence of states $(c_i)_{i=j}^n$ with the maximum likelihood, as well as that probability $P_{j,c'}^{\leftarrow}$.

If we have run Viterbi's algorithm forwards and backwards and obtained both dynamic programming arrays, and we wish to compute the maximum likelihood chord

sequence with the additional constraint that the chord at some location i is constrained to the chord c^*, but while also obeying all chord locks at positions other than i, we can simply compute it as

$$\frac{P^{\rightarrow}_{j,c^*} \cdot P^{\leftarrow}_{j,c^*}}{e_{j,c^*}},$$

using three quantities we have already computed and taking constant extra time. (Note in particular that we can perform this computation even if the "main" Viterbi algorithm has a different chord locked at position j. This is the reason we handle locked chords in the Viterbi algorithm by ignoring all other possible chords in the step after the locked chord, rather than by just setting the likelihoods of all chords other than the locked one at that position to 0, as MySong proposes, even though the "main" maximum likelihood sequence of chords would be the same with either approach.)

There is also a symmetry observation that can make computing these probabilities slightly easier. In general, if we only consider chord transitions, the likelihood of a sequence of chords (c_i) in our HMM is just

$$P(C_1 = c_1) \cdot \prod_{i=1}^{n-1} P(C_{i+1} = c_{i+1} \mid C_i = c_i).$$

If we derive these probabilities directly from the maximum likelihood estimate, we have that

$$\begin{aligned} P(C_1 = c_1) \cdot \prod_{i=1}^{n-1} P(C_{i+1} = c_{i+1} \mid C_i = c_i) &= P(C_1 = c_1) \cdot \prod_{i=1}^{n-1} \frac{P(C_i = c_i \cap C_{i+1} = c_{i+1})}{P(C_i = c_i)} \\ &= \frac{\prod_{i=1}^{n-1} P(C_i = c_i \cap C_{i+1} = c_{i+1})}{\prod_{i=2}^{n-1} P(C_i = c_i)}. \end{aligned}$$

This rewriting is forwards-backwards symmetric and makes the implementation of the forwards and backwards Viterbi's algorithm slightly easier. However, note that we are glossing over the "end" transitions where c_n is the last chord in a sequence in making this substitution. Fortunately, the chord progressions in our data are long

enough that this does not significantly affect the model's recommendations.

4.4 Architecture

RiffShuffle is a single-page web application built with React.js that communicates over a websocket with a Python server. For simplicity, the backend server is fully stateless; the client sends the full state of the melody and chords at each step. This backend stores the data described in the previous section, which is pre-summarized into transition matrices and probability distributions and then serialized into a short form that could be easily loaded. The server loads this precomputed HMM and uses it to harmonize each melody it receives, while obeying any locked chords in the input.

This architecture was largely set up so that the Python server would have easy access to popular machine learning libraries and models, but give the simplicity of the model we ended up using, it may not be necessary. Future work may look towards improving the performance of this architecture.

4.5 Interface

RiffShuffle's interface is depicted in Figure 4-1. To start using RiffShuffle, users can upload a MIDI file in the very top section A. Note that RiffShuffle does not currently allow the user to edit the melody from inside the tool, even though inputting a melody is an important part of using any composition tool in practice. We instead would generally expect users to construct a MIDI in a different tool and upload it, to keep our tool simple. This also facilitates interoperability, since MIDI-editing tools are very powerful and well-established. For the user study, they are also given the option of choosing a melody from a dropdown. There is also a popup where users can import and export their data as a simple JSON file; this includes the melody and chords but also other settings, such as the parameters described in Subsection 4.5.1 below.

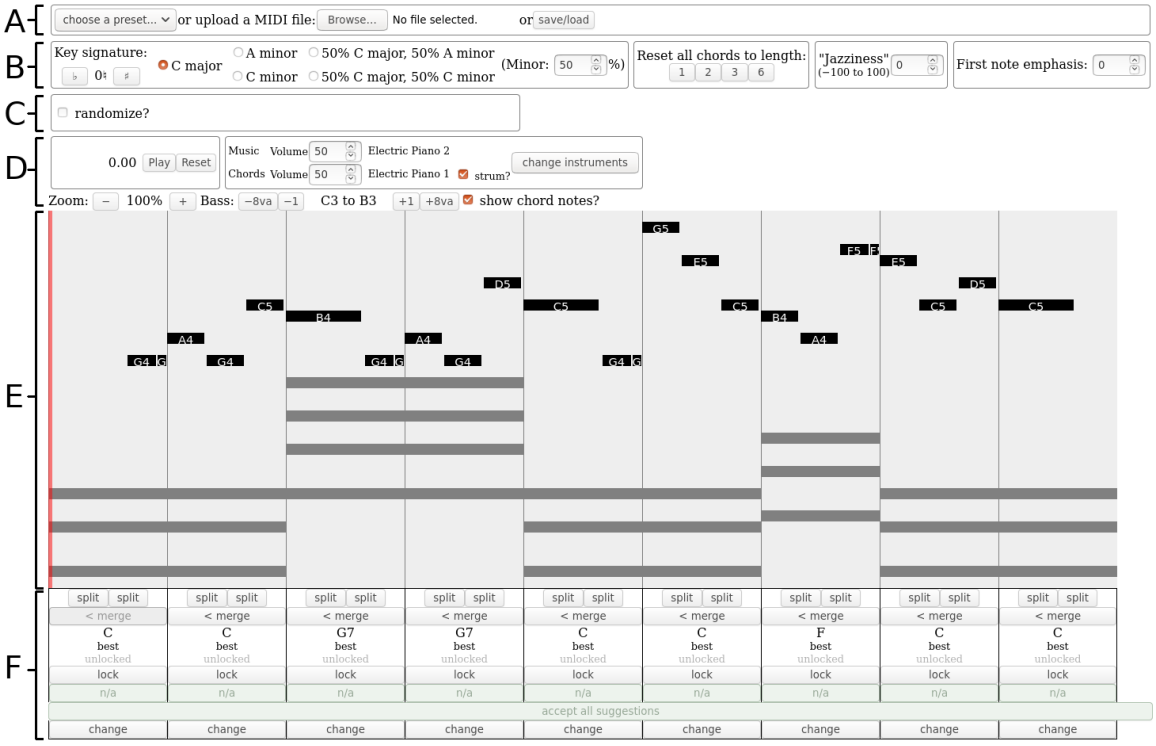

Figure 4-1: Screenshot of RiffShuffle's full interface. A: Song upload/choice and import/export. B: global settings. C: randomization. D: playback settings. E: melody and chord display. F: chord adjustment.

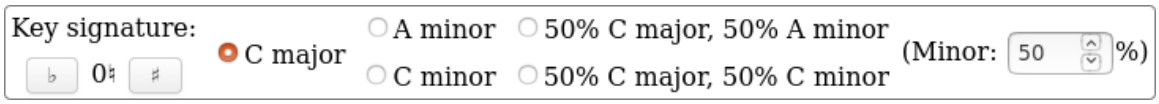

Figure 4-2: Screenshot of RiffShuffle's key/mode selection interface.

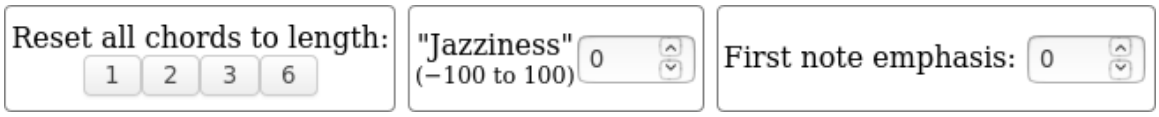

Figure 4-3: Screenshot of RiffShuffle's global settings selection interface.

4.5.1 Global Model Settings

The next part of the interface (part B of Figure 4-1) consists of several global settings, which control parameters of the model that affect all generated chords. The interface uses metadata from imported MIDI files, namely the key signature and time signature, as defaults for these settings.

The key signature and mode are exposed as a direct option, as in Figure 4-2, and RiffShuffle does not attempt to guess it based on the input melody and chords since it is a field encoded in MIDI, and we expect that users will generally not have trouble identifying the desired tonic and key signature of a melody they wish to harmonize. If they were composing a melody, they would likely have picked a tonic and key already. If the user chooses one of the modes that combines major and minor, the minor percentage corresponds to the "happy factor" described in Subsection 4.3.1.

The rest of the global settings affect the frequency of chord transitions globally, as well as the "jazz factor" and first note emphasis parameter that were also described in Subsection 4.3.1. The time signature and tempo cannot be changed, but the frequency of chord transitions can be set globally here or fine-tuned later. The displayed jazziness, which can range from -100 to 100, is divided by 100 to obtain the internal jazz factor. The displayed first note emphasis, which can range from 0 to 100, is converted somewhat arbitrarily from value x to internal parameter $\exp(8x/100)$, so that it takes on values from 1 to roughly 2981, the latter number of which is easily enough to allow any first note in a measure to overwhelm any non-first notes. Both settings have explanations for users that display in a tooltip if the user hovers over

Figure 4-4: Screenshot of RiffShuffle's randomization feature.

them.

4.5.2 Randomization

We also have an optional randomization feature (part C of Figure 4-1, depicted active in Figure 4-4), in which the interface no longer tries to compute the optimal (maximum likelihood) chord sequence, but simply samples a random chord sequence as described in Subsection 4.3.2. Again, a crucial assumption of our project is that a given melody has many reasonable harmonizations and different users will prefer different chord sequences. A randomize button is a low-effort way for users to expose themselves to alternative chord progressions. For any given random chord sequence, most people might consider it less reasonable than the default maximum likelihood chord sequence, but users can try to find one in which they are in the minority that prefers it. If changing a slider takes considerably less thinking, effort, and skill than choosing chords oneself, repeatedly pressing the randomize button requires even less thinking.

The button uses a random seed so that the randomness is deterministic, making previous random chord progressions recoverable. This way, users can save a seed they like and return to it later. This would also make it relatively easy to provide a simple way to return to a previous randomization, though we unfortunately lacked the time to implement this.

As expected, the randomization feature also exposes the chaos/determinism factor described in Subsection 4.3.2. Here the display value, which ranges from -100 to 100, is again somewhat arbitrarily converted from value x to internal parameter $r = (1 - x/100)^3$. The important requirements are that $r = 1$ when $x = 0$ and $r = 0$ when $x = -100$. We decided to choose an interpolation function that makes $r = 8$ when $x = 100$, which is usually enough for larger probabilities to completely overwhelm

smaller probabilities and make the process effectively deterministic.

4.5.3 Playback

The playback settings (part D of Figure 4-1) allow users to choose which instruments they want to hear, and zoom in and out horizontally. Of course, users can also play and stop, adjust the volume of the melody and chords, and click in the display (part E) to seek. The display can be scrolled horizontally with standard browser scrolling.

4.5.4 Chord Adjustment

Unlike MySong, we allow chords to vary in duration, as this is very common; even "Happy Birthday" notably contains a short I-V-I chord transition. Instead of trying to make the model determine the chord durations as well, however, we give users full control over when chords start and end through simple "split" and "merge" buttons, depicted on the top of Figure 4-5, which split a chord into two and merge a chord with its predecessor. Note that our model does not treat chords differently based on duration. Every chord transition and every observation of a note during a chord is given equal weight regardless of how long the chord lasts.

4.5.5 Specific Chord Choices

In addition, as in MySong, we allow users to change specific chords to other values and "lock" them, as displayed in Figure 4-5. That is, they can choose any of the first recommended chords, or the list of all available chords in a popup, and the model will always obey locked chords.

However, unlike in MySong, locking chords causes nearby chords to be re-suggested. In the first iteration, whenever a user locks any chords, this caused unlocked chords to automatically recompute to the optimal values, but users found these changes unexpected and confusing. In the second iteration, when locking/unlocking chords causes RiffShuffle's suggestions to change, the suggestions are not automatically taken but instead displayed in the green buttons below the current chords. Users must click

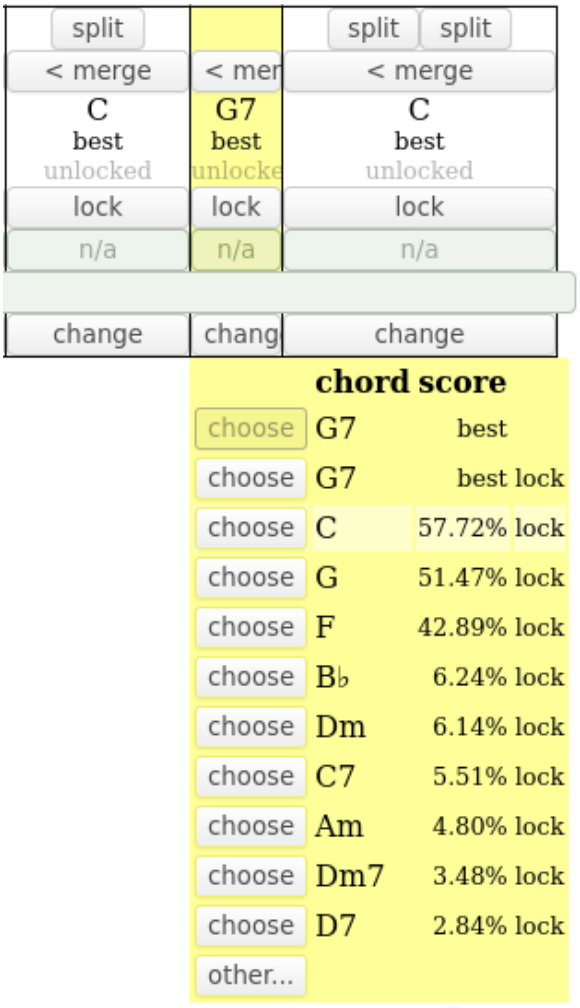

Figure 4-5: Screenshot of RiffShuffle's chord splitting, merging, choice, and locking features.

each green button to accept each suggestion, or they can click a long green button to accept all suggestions.

Finally, the top 10 recommended chords at each slot receive a percentage score, which is the relative likelihood of the optimal chord progression using that chord compared to the likelihood of the overall chord progression. These values are computed using the results of the bidirectional Viterbi algorithm described in Section 4.3.3, and can give users a better quantitative sense of how plausible the model thinks these runner-up recommendation chords are. This can give users guidance on deciding whether alternative chords in this position are worth considering, or whether the user's attention might be better spent changing chords elsewhere.

Chapter 5

Results and Evaluation

To evaluate our system, we recruited online study participants to test it and provide their harmonizations as well as subjective ratings. To reduce the scope of the user task, we chose several fixed test melodies that we asked test subjects to harmonize. Our test melodies were chosen as two popular folk songs from an online database that were not in the Nottingham Database, as well as two of the highest-ranked rock songs from the Rolling Stones' List of 500 Songs that were not analyzed by Temperley and deClercq. These songs were not chosen fully objectively; we made some effort to choose songs with relatively simple melodies, and among the folk songs preferred songs in major rather than minor key since our training data had substantially more major key songs. Nevertheless, we tried to limit the human influence on the song choice, and in particular we did not look at RiffShuffle's recommendations on these songs' melodies while selecting them. We therefore believe user studies using the songs we selected are at least somewhat representative of RiffShuffle's performance in general.

In general, flaws in RiffShuffle's default generated chords for all melodies were quite evident, so we expected that participants would not rate its output with no human intervention highly, particularly if the participants had more musical background. Nor would RiffShuffle necessarily suggest chords that are better than what participants with the best musical background could independently come up with. We hypothesized that the aspect of RiffShuffle that would receive the highest ratings

would be its ability to make harmonization more efficient.

5.1 First User Study

For the first test, 7 participants were recruited through various channels and asked to harmonize the melodies and fill out some survey questions. To reduce the burden of participation, we allowed participants to finish the survey after harmonizing anywhere from one to all four of the melodiies. All participants completed at least the first two melodies, and 3 participants harmonized all four, for a total of 20 participant-song harmonizations. Not all participants were willing to share their harmonizations (those participants simply provided ratings), but the harmonizations we do have are striking. A small selection of RiffShuffle's default harmonization as well as the harmonizations generated by six participants on the first test song ("Save the Last Dance for Me" by The Drifters, 1960, which is #182 on Rolling Stones' list) is shown in Figure 5-1.

In the example, we see that by default, RiffShuffle harmonizes the tied F♯ with one V7 but ignores it for the rest of the fifth and sixth bars, considering it more important to return to the tonic chord, even though this does not correspond to a resolved part of the song. More interesting is the fact that all participants' harmonizations diverged within the first six bars of the first song, despite the fairly unremarkable melody that mostly consists of the third scale degree G♯. The tonic chord was unsurprisingly popular, but two participants somewhat creatively put a secondary dominant tonicizing the ii chord in the transition from bar 4 to 5, even though by default both the secondary dominant nor the ii chord itself were only the 3rd most likely chords in their respective positions suggested by RiffShuffle.

Of course, one cannot attribute the creative chords to RiffShuffle's interaction; it may well be that the participants would be equally or more creative harmonizing this directly on paper or in a scorewriter that made no suggestions whatsoever. However, this extent of divergence is encouraging in that it suggests RiffShuffle does facilitate interaction and experimentation, even by first-time users, instead of being perceived as a didactic chord-generation tool.

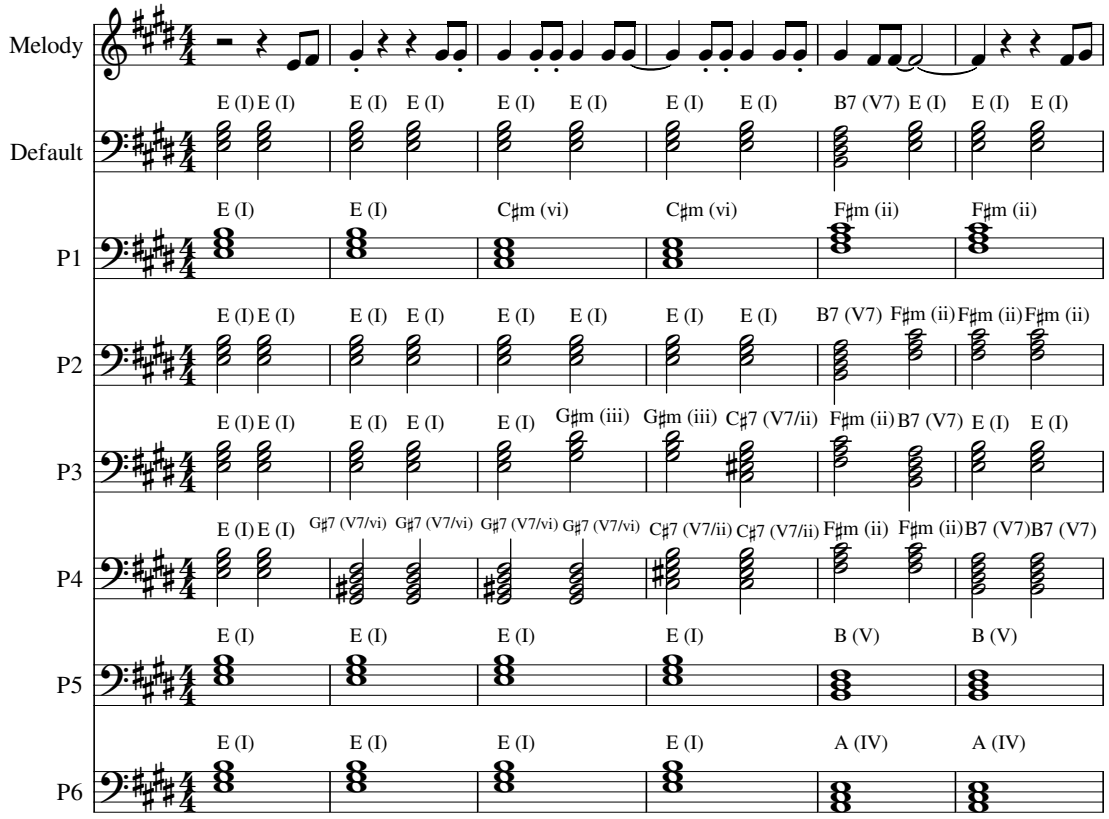

Figure 5-1: The first six bars of the first six participants' harmonizations of the first test song.

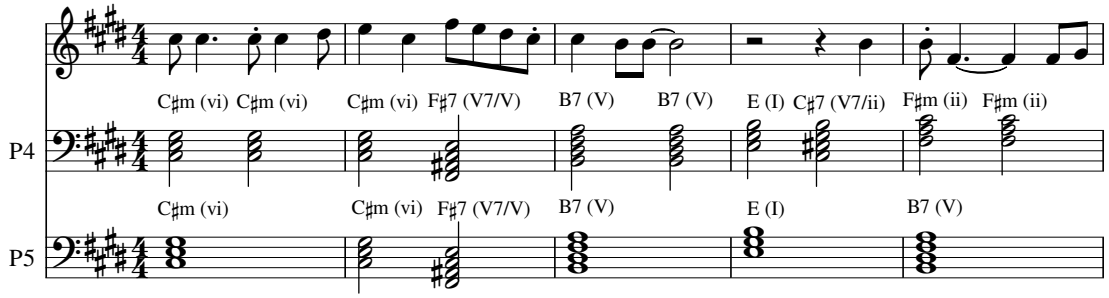

Figure 5-2: Some interesting features of harmonizations.

Two participants locked most chords, while one participant locked 14 (about 40% of the chords) and four participants locked no more than 8. Participants submitted some other interesting chords that were not suggested by default, including several more secondary dominants. For example, in Figure 5-2, one may observe that Participant 4 added more secondary dominants in the later part of the song, which is consistent with their harmonization of the first few measures. Again we do not suggest to what extent RiffShuffle prompted or inspired these changes, as these may well be the same chords Participant 4 would have generated through traditional manners; but it supports the case that users are able to bring their own styles and preferences while using the tool. Figure 5-2 also shows Participant 5's successful use of the chord-splitting interface to insert a brief secondary dominant. All of Participant 5's other chords in this harmonization lasted for a full measure.

Similar patterns can be observed from other test songs. Figure 5-3 shows the first six bars of the harmonizations of the second test song. (Participant 3 uploaded the wrong file to the survey, so their harmonization was lost.) The differences are less drastic than before, but all participants modified RiffShuffle's rather boring default of entirely I chords. As a further demonstration of users' ability to use the chord merging and splitting feature, Figure 5-4 shows the last four bars of another test song and all the harmonizations we obtained. Two participants split the penultimate measure into multiple chords to add a V7-I resolution.

Table 5.1 shows the survey results for all 20 user-song harmonizations. The sample is of course too small to draw significant conclusions, but there were some trends consistent with our hypothesis and other expectations. The statement people agreed with the most strongly was by far the last statement, that the interface made chord generation faster, as hypothesized; furthermore, no participant chose an option on the Disagree side, and only one person remained neutral. Participants also rated their generated chord progressions as noticeably improved over the default progression.

In unstructured feedback, one feature that stood out as being positively received was the randomization feature. One participant wrote, "I would say the best feature is the randomization feature, since it can take the chords in direction that an amateur

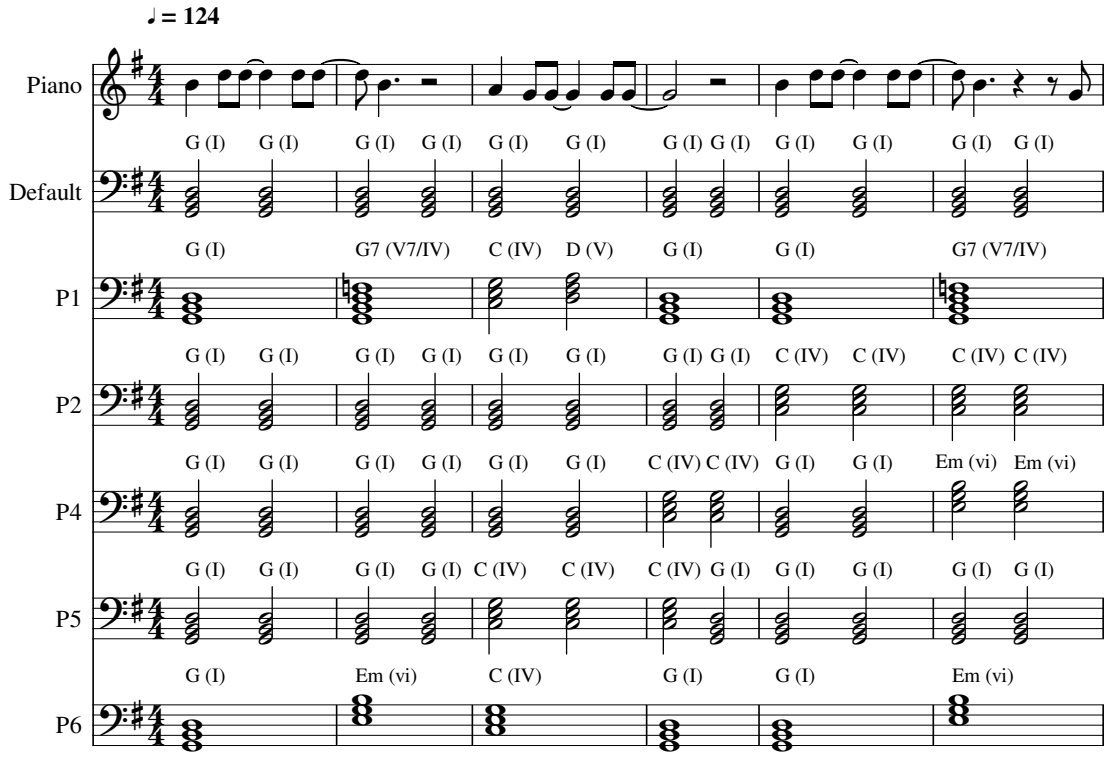

Figure 5-3: The first six bars of harmonizations of the second test song.

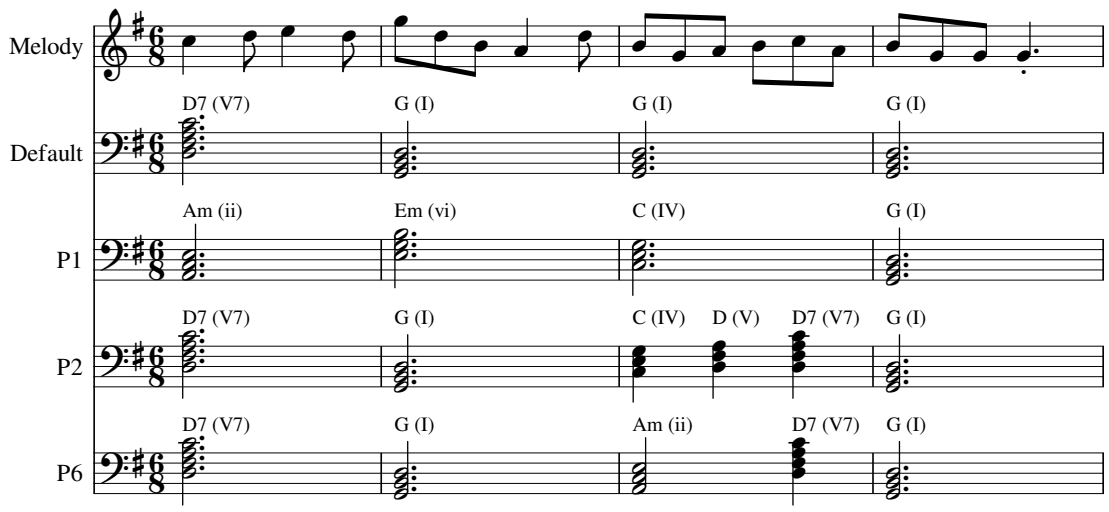

Figure 5-4: The last four bars of harmonizations of another test song.

Question	1	2	3	4	5	Avg
"The default progression... sounds good."	1	8	4	6	1	2.90
"The default progression... sounds creative."	5	9	2	4	0	2.25
"The chord progression I generated sounds good."	0	0	2	15	3	4.05
"The chord progression I generated sounds creative."	0	3	2	10	5	3.85
"The chord progression I generated reflects my preferences."	0	1	2	10	7	4.15
"The interface allowed me to generate better progressions...."	0	1	3	7	9	4.20
"The interface allowed me to generate good-sounding progressions more quickly...."	0	0	1	7	12	4.55

Table 5.1: The ratings received for all harmonizations in the first user study. Each table entry indicates the number of participants who responded with the column label for that row's question, with 1 being Strongly Disagree and 5 being Strongly Agree.

might not think of, and then they sometimes end up sounding good anyway!" On the other hand, participants expressed annoyance that chords automatically changed surrounding chords. Three participants mentioned that they wanted to use minor 7th chords and diminished chords.

Based on these results of this user study, we modified the system to not automatically change chords in response to local chord locking and selection, but merely suggest the recomputed chords in a way that users could easily accept. We also added support for minor seventh chords and diminished chords.

5.2 Second User Study

To focus evaluation on the questions we are the most interested in and further reduce the burden of taking the user study, we reduced the number of questions we asked in each harmonization and conducted a second user study. Unfortunately we were only able to recruit 6 users, who completed 13 harmonizations in total.

Again, we display five participants' harmonizations of the first test song in Figure 5-5. Although we used E major as the key for all Roman numeral analyses for consistency, we see that in this user study, Participants 1 and 2 leaned heavily into the relative minor key of C♯ minor for their chords, giving the song a very different

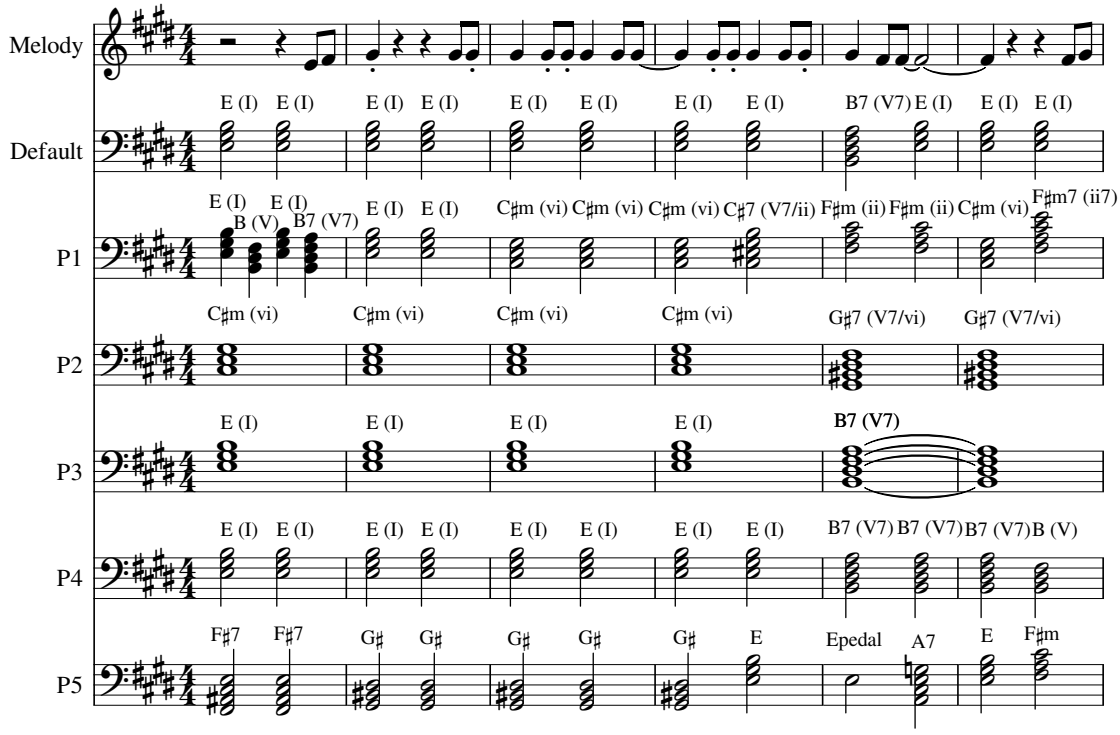

Figure 5-5: The first six bars of five participants' harmonizations of the first test song, in the second user study.

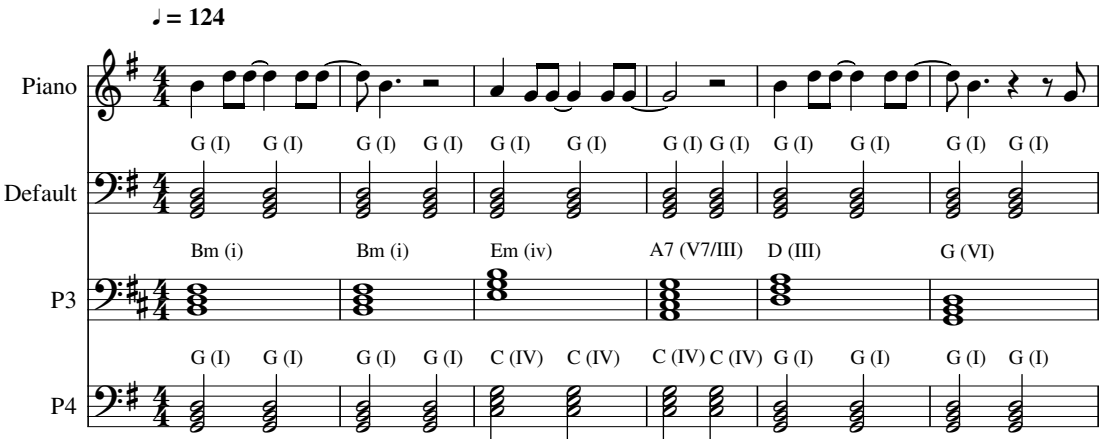

Figure 5-6: Some harmonizations of the second test song, in the second user study.

feel, and it may make more sense to analyze them in the minor key. Also, the last participant adjusted jazziness to 85% and moved the chords up to be played much above the melody, making traditional analysis difficult.

As another example, Figure 5-6 shows the first six bars of harmonizations of the second test song. (Note that we reordered the songs in the second user study to gather a greater variety of examples, so fewer participants in this study got to this song.) P4's harmonization is fairly normal, but P3 arrived at a highly unusual chord progression by harmonizing the melody as a blend of D major and B minor. This is somewhat plausible because the incomplete melody fragment we presented ends on a B, and for the same reason we thought it best analyzed in B minor. This example shows how a user can produce a radically different chord progression in RiffShuffle with relatively little effort.

We will look at one final batch of example harmonizations, Figure 5-7. Even though the song was initially loaded in F major, all participants harmonized the melody in chords that made the most sense in D minor, either by changing the system's mode or simply by choosing D minor chords directly. In addition, we can observe that one participant used an unusual diminished chord, weakly demonstrating the usefulness of adding it to the range of chords that RiffShuffle can generate, although it might be better analyzed as a passing tone. Future systems might con-

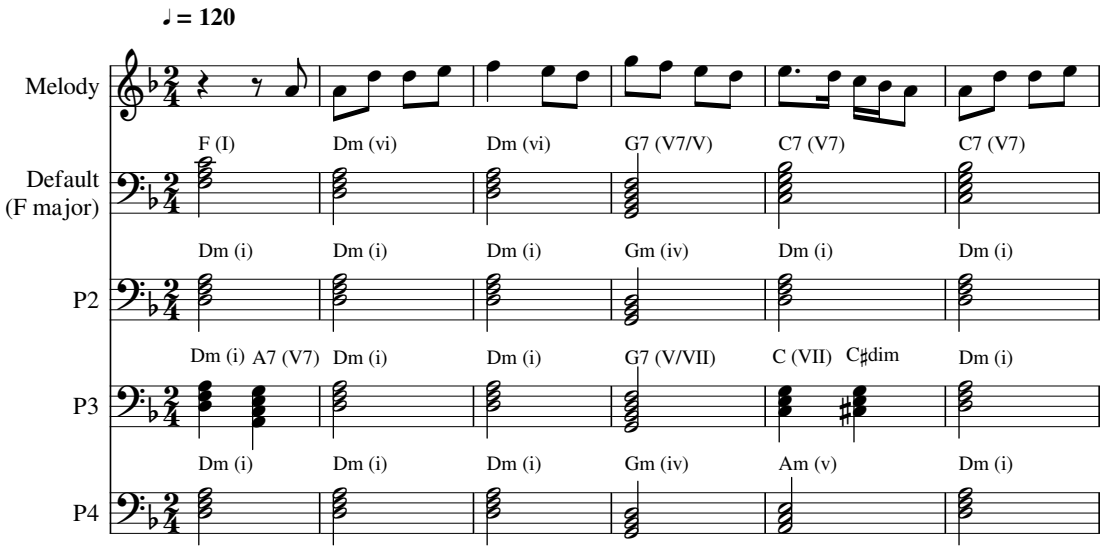

Figure 5-7: Some harmonizations of another test song, in the second user study.

Question	1	2	3	4	5	Avg
"The chord progression I generated reflects my preferences."	0	0	0	10	3	4.23
"The interface allowed me to generate better progressions. . . ."	1	3	0	4	5	3.69
"The interface allowed me to generate good-sounding progressions more quickly. . . ."	1	2	0	1	9	4.15

Table 5.2: The ratings received for all harmonizations in the second user study. Each table entry indicates the number of participants who responded with the column label for that row's question, with 1 being Strongly Disagree and 5 being Strongly Agree.

sider supporting features such as passing tones explicitly during chord progression recommendation.

Table 5.2 shows the survey results for these user-song harmonizations. These results are weakly inconsistent with our hypothesis; this group of users agreed more that the chord progression reflected their preferences than that the interface assisted them in arriving at it.

However, in unstructured feedback, participants again expressed that they liked the randomization feature. One participant wrote, "I knew the melody so the chords

that 'felt right' were the ones I was having trouble breaking away from, and the randomize button gave me some new ideas." Another wrote, "Went and tested out randomization for this one. Surprisingly helpful! It found a few interesting chord [sic] that I would never have thought to put and it doesn't sound half bad."

5.3 Conclusions

Overall, through our survey results we believe that users generally enjoyed using RiffShuffle. Although it depends on the specific user, most users were able to use it in ways that, at least according to them, reflected their preferences and generated better chords more efficiently. The biggest surprise was the extent to which users singled out the randomization feature for praise. A more general takeaway might be that easy access to randomization is a good feature in interactive systems that are meant to foster or emulate a creative process.

Chapter 6

Future Work

Although we believe RiffShuffle shows the promise of a more interactive paradigm for automatic harmonization, it is a fairly simplistic system and model. Below, we discuss some possible directions for future work.

6.1 Improved Models

Perhaps the most obvious area of future research is simply to adapt one of the more advanced machine learning models discussed in Section 3.2. The reasons we chose the simple Hidden Markov Model were outlined in Section 4.3, and we would expect that additional effort and performance engineering might be necessary to adapt more advanced models, but these obstacles are not insurmountable and would likely produce higher-quality chords.

In addition, one question that seems relatively unexplored in the literature is to find a way to make a model support a very large variety of chords despite limited training data, and circumvent the need for chord simplification as discussed in Subsection 4.2.3. The most important reason we perform chord simplification is that because without it, many chords would appear far too rarely in the training data for a model to effectively learn proper usage. However, many chords have similar function and work in similar contexts, in ways that can probably be mechanically described or approximated (e.g. inversions of a dominant chord). It might be interesting to

see if a model can use a small additional amount of musical knowledge about what chords are similar to learn from rare occurrences of chords and generate them where appropriate. The downsides of potentially generating an implausible chord is less in the interactive paradigm, since users can always change the chord or reject suggestions. A secondary reason we perform chord simplification is that this makes the Viterbi algorithm's performance tractable, but there are optimizations and heuristics available for improving its runtime, so this obstacle should also be surmountable.

6.2 Improved or Varied Input Data

As discussed in Section 4.2, we synthesized three sources to create a training corpus of roughly 3,500 songs, but this is still a fairly small number. HTPD3 [18] is an example of a larger training set. Increasing the data set size should also improve the quality of the generated chords.

Another idea briefly mentioned in Section 4.2 that was not implemented in RiffShuffle is the idea of interactively interpolating between different data sets. Users could choose between data sets with distinct styles to power the model's suggestions. This is not very far from the way in which we partitioned our input data and interpolated between minor and major scales with a slider. This effect might be most interesting if we could find multiple data sets with highly distinctive styles; one plausible choice that is also popular in the literature would be the set of Bach chorales, which is also seen in [8], because the style follows mostly rigid rules and would probably feature very different chords and chord transitions from other data sets.

6.3 Additional Interactions

RiffShuffle focuses on the harmonization aspect of composition, and assumes the melody as fixed while trying to find a chord progression. This is not a particularly realistic approach, since a composer can freely change either the melody or chords

being composed at will. It might be interesting to design a system where users can interactively change either the melody or the chords, and the system can make suggestions for both.

Also, RiffShuffle has focused on global settings like changing the key, mode, and some global model parameters like "jazziness", as well as highly local settings like individual chords. It might be interesting to try to find a middle ground of interaction where users can change parameters for ranges of chords or melody notes that are longer than individual chords but shorter than the whole piece.

Yet another consideration is to what extent an interactive tool could also be pedagogical, and teach the user ideas or general guidelines about composition that the user could apply even outside the system. This might involve making the model reveal more about why each suggestion was made, either in terms of the model's internals in a way that the user can extrapolate from, or in terms of external formal rules that analyze the function of the suggested chord. One could also integrate a purely rule-based symbolic recommendation engine that is presented side by side with the machine learning model and allow users to learn about the similarities and differences.

Finally, RiffShuffle's analysis is performed offline, and one might consider how much of its design could be adapted for online analysis, which harmonizes a melody as it is inputted live somehow.

6.4 Additional Evaluation

One simple area of possible improvement is just to run larger user studies, so that we can get a more reliable assessment of how users use our system and what they think of fit. We might also hope to try running user studies with different variants of the interface to compare their performance and the subjective ratings they receive.

There are many evaluation strategies we also did not explore. We could ask human judges to listen to other people's compositions and rate them. One question we could investigate, which would involve many more subject participants than were available,

is whether users have consistent styles that end up transferring between pieces. One way we could imagine testing this is asking many users to harmonize many pieces and then asking other judges to listen to the harmonizations and see whether they consistently rank some users' compositions over other users', or even identify distinct users' compositions.

Appendix A

Tables

Table A.1: Correlation between chord vectors in the RS 200 corpus with absolute value > 0.35. (Chord vectors are the result of, for each chord, annotating every song with 1 if it has that chord and 0 if not.)

Chord 1	Chord 2	Correlation
I	i	−0.810
♯iv	VII	0.663
♯iv	vii	0.621
i	♭VI	0.611
I	IV	0.610
I	♭VI	−0.508
♭ii	VII	0.496
♭III	♭VII	0.475
i	IV	−0.473
♭III	♭VI	0.459
♭VI	♭VII	0.459
IV	iv	−0.457
ii	iii	0.454
♯iv	♭vi	0.443
♭ii	♯iv	0.443
IV	V	0.436
I	♭III	−0.435
I	V	0.428
i	iv	0.423
I	iv	−0.413
♭II	♯iv	0.404
i	♭III	0.402
iv	♭VI	0.392
i	♭VII	0.383
♭III	iv	0.376
II	vi	0.374
iii	vi	0.357
III	vi	0.357
iv	v	0.351

Table A.2: Chords by frequency in MARG's data set (with some subjective normalization). Chord appearances in separate measures were considered separate.

Chord	Frequency
major	42478
dominant 7th	20134
minor	12824
minor 7th	7454
major 7th	3091
major 6th	1461
no chord	1257
dominant 9th	1210
minor 6th	713
diminished	673
half-diminished 7th	589
suspended 4th	585
augmented 7th	302
minor 9th	302
major 9th	225
augmented	224
dominant 13th	212
power (no 3rd)	104
suspended 2nd	86
dominant 11th	52
minor 11th	35
major 6th and 9th	29
minor-major 7th	24
augmented 9th	12
pedal	12
diminished 7th	4
minor 7th flat 5th	3
minor 9th	3
minor 13th	2
dominant 7th suspended 4th	1

Table A.3: Chords by frequency in all transcriptions in the 200-song Rock Corpus. Chord appearances in separate measures were considered separate. Note that this counts both authors' transcriptions.

Chord	Frequency
major	35192
minor	10047
dominant 7th	1309
minor 7th	1286
major, 1st inversion	950
major, 2nd inversion	706
major 7th	228
minor, 1st inversion	190
suspended 4th	137
major 11th	136
augmented 11th	67
minor 7th, 1st inversion	64
half-diminished 7th	63
dominant 9th	62
major 9th	61
major sharp 9th	55
minor 7th with suspended 4th	52
diminished	44
dominant 7th, 1st inversion	41
diminished 7th	36
half-diminished 7th, 2nd inversion	34
minor 9th	33
dominant 7th, 2nd inversion	22
dominant 7th sharp 9th	21
dominant 7th, 3rd inversion	21
diminished 7th, 3rd inversion	15
minor 7th, 2nd inversion	14
dominant 7th with suspended 4th	12
major flat 5	12
minor 7th, 3rd inversion	12
diminished, 1st inversion	11
major 7th, 1st inversion	10
half-diminished 7th, 1st inversion	9
half-diminished 7th, 3rd inversion	8
minor 11th	8
dominant 7th flat 9th	6
diminished 7th, 2rd inversion	5
augmented	4
augmented 7th, 1st inversion	3
augmented 9th	3
augmented 7th	2
major 7th, 3rd inversion	1

Table A.4: Chords by frequency in the Nottingham Music Database.

Chord	Frequency
major	16244
dominant 7th	5720
minor	4245
major, 1st inversion	235
dominant 7th, 2nd inversion	76
major, 2nd inversion	60
dominant 7th, 1st inversion	53
minor, 1st inversion	26
minor, 2nd inversion	15
dominant 7th, 3rd inversion	13
minor, minor 7th in bass	12
minor 7th	9
major 6th	9
no chord	5
minor 7th, 3rd inversion	4
minor 6th	3
dominant 7th flat 9th	3
minor, major 7th in bass	2
minor, major 6th in bass	2
dominant 7th, major 6th in bass	2
major, major 7th in bass	1
major, 4th in bass	1

Bibliography

[1] AWS DeepComposer. Amazon. Accessed April 30, 2020. `https://aws.amazon.com/deepcomposer/`

[2] Cuthbert, Michael Scott, and Christopher Ariza. "`music21`: A Toolkit for Computer-Aided Musicology and Symbolic Music Data." 11th International Society for Music Information Retrieval Conference (ISMIR 2010). Accessed 12 May 2020. `https://web.mit.edu/music21/`

[3] de Clercq, Trevor, and David Temperley, "A Corpus Analysis of Rock Harmony". Popular Music 30 [2011], 47–70. 2011. Accessed 29 September 2019. `http://rockcorpus.midside.com/2011_paper/declercq_temperley_2011.pdf`

[4] Bishop, Christopher M. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] Dhariwal, Prafulla, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, Ilya Sutskever. "Jukebox: A Generative Model for Music". OpenAI. Accessed 30 April 2020.

[6] Groves, Ryan. "Automatic Harmonization Using a Hidden Semi-Markov Model." Musical Metacreation: Papers from the 2013 AIIDE Workshop (WS-13-22). Accessed 27 February 2019.

[7] Huang, Cheng-Zhi Anna, David Duvenaud, and Krzysztof Z. Gajos. "ChordRipple: Recommending Chords to Help Novice Composers Go Beyond the Ordinary." *IUI 2016*, March 7–10, 2016. Accessed 12 September 2019.

[8] Huang, Cheng-Zhi Anna, Curtis Hawthorne, Adam Roberts, Monica Dinculescu, James Wexler, Leon Hong, and Jacob Howcroft. "The Bach Doodle: Approachable music composition with machine learning at scale", 20th International Society for Music Information Retrieval Conference, Delft, The Netherlands, 2019. Accessed 2 May 2020.

[9] Lim, Hyungul, Seungyeon Rhuy, and Kyogu Lee. "Chord Generation from Symbolic Melody Using BLSTM Networks." 18th International Society for Music Information Retrieval Conference (ISMIR 2017). Accessed 29 September 2019. `https://arxiv.org/abs/1712.01011`.

[10] Huang, Cheng-Zhi Anna and Vaswani, Ashish and Uszkoreit, Jakob and Shazeer, Noam and Hawthorne, Curtis and Dai, Andrew M and Hoffman, Matthew D and Eck, Douglas. "Music Transformer: Generating Music with Long-Term Structure." arXiv preprint arXiv:1809.04281, 2018. Accessed 30 April 2020.

[11] Makris, Dimos, Ioannis Karydis, and Spyros Sioutas. "Automatic Melodic Harmonization: An overview, challenges and future directions." 2016. *Trends in Music Information Seeking, Behavior, and Retrieval for Creativity*, IGI Global. 2016. Accessed 1 January 2019.

[12] Merryman, Marjorie. *The Music Theory Handbook.* Boston: Schirmer, 1997.

[13] Morris, Dan, Ian Simon, and Sumit Basu. "Exposing Parameters of a Trained Dynamic Model for Interactive Music Creation." AAAI'08 Proceedings of the 23rd national conference on Artificial intelligence — Volume 2. July 2008. Accessed 12 September 2019.

[14] Pasquier, Philippe, Arne Eigenfeldt, Oliver Bown, and Shlomo Dubnov. "An Introduction to Musical Metacreation." Computers in Entertainment (CIE) — Special Issue on Musical Metacreation, Part I, Volume 14 Issue 2, Summer 2016. Accessed 9 September 2019. `https://dl.acm.org/citation.cfm?id=2930672`.

[15] Payne, Christine. "MuseNet." OpenAI, 25 April 2019, `openai.com/blog/musenet`. Accessed 30 April 2020.

[16] Simon, Ian, Dan Morris, and Sumit Basu. "MySong: Automatic Accompaniment Generation for Vocal Melodies." CHI '08 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. April 2008. Accessed 12 September 2019.

[17] Tsushima, Hiroaki, Eita Nakamura, Katsutoshi Itoyama, and Kazuyoshi Yoshii. "Interactive Arrangement of Chords and Melodies Based on a Tree-Structured Generative Model." 19th International Society for Music Information Retrieval Conference, Paris, France, 2018. Accessed 9 September 2019.

[18] Yeh, Yin-Cheng, Wen-Yi Hsiao, Satoru Fukayama, Tetsuro Kitahara, Benjamin Genchel, Hao-Min Liu, Hao-Wen Dong, Yian Chen, Terence Leong, and Yi-Hsuan Yang. "Automatic Melody Harmonization with Triad Chords: A Comparative Study." arXiv:2001.02360 [cs.SD] 2020. Accessed 12 February 2020.