Judah Daniels

Inferring Harmony from Free Polyphony

Computer Science Tripos – Part II

Clare College

July, 2023

Declaration of originality

I, Judah Daniels of Clare College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. I am content for my dissertation to be made available to the students and staff of the University.

Signed Judah Daniels

Date April 19, 2023

Proforma

Candidate Number:	**2200D**
Project Title:	**Inferring Harmony from Free Polyphony**
Examination:	**Computer Science Tripos – Part II, July, 2023**
Word Count:	**5724**[1]
Code Line Count:	**2272**[2]
Project Originator:	**Christoph Finkensiep**
Supervisor:	**Dr Peter Harrison**

Original Aims of the Project

Work Completed

Special Difficulties

There were no special difficulties encountered in this project

[1]This word count was computed by `texcount main.tex`
[2]This code line count was computed by using `cloc`

Contents

List of Figures

Acknowledgements

I'd like to thank Christoph Finkensiep and Peter Harrison for being awesome.

Introduction

1.1 Motivation and Aim

Most of western tonal music can be described using a sequence of chords, representing a higher level harmonic structure of a piece. Automatic Chord Estimation (ACE) is the task of inferring the sequence of chords for a given piece from symbolic data. There is a small, finite set of chord types, but each chord can be realised on the musical surface in a practically infinite number of ways. Given a score S (a symbolic representation of a piece of music), we wish to infer the sequence of underlying chord types $L = l_0, \ldots, l_n$.

$$\hat{L} = \arg\max_{L} p(L|S) \tag{1.1}$$

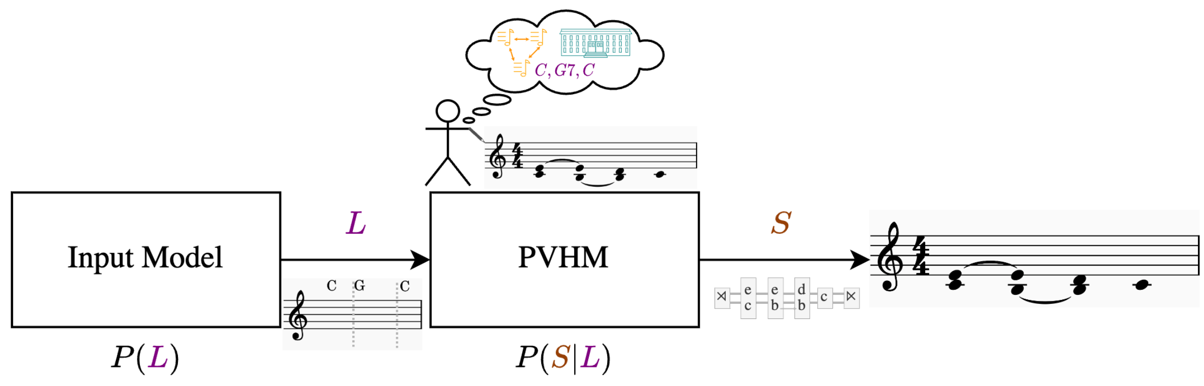
We present a **novel solution**, the **ProtoVoice Harmony Model (PVHM)** that infers the sequence of chord types given score using probabilistic models of latent structure and harmony, dimensionality reduction, parsing strategies and heuristic search methods.

Automatic Chord Estimation has both theoretical and practical applications. Analysis of music often starts with the manual labeling each chord, which is a time consuming and cogntively demanding expert task. Sequences of chords provide compact representations for use in analysis, music identification and music similarity finding. More broadly speaking, any system that involves the understanding of written tonal music will benefit from chord estimation.

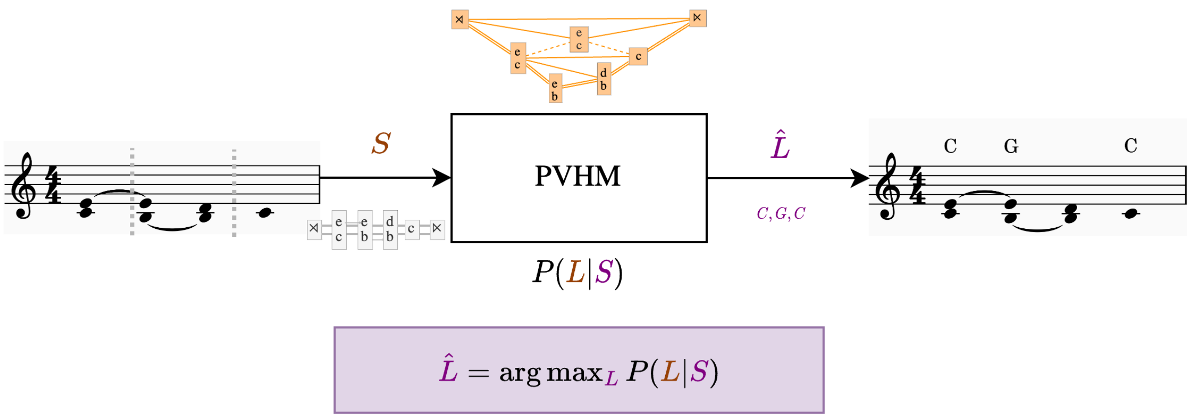
The paper *Modeling and Inferring Protovoice Structure in Free Polyphony* describes a generative model that encodes the recursive and hierarchical dependencies between notes, giving rise to a grammar-like hierarchical system [10]. This model can be used to reduce a piece into a hierarchical structure which encodes an understanding of the tonal/harmonic relations.

While the original model could in theory be used to generate harmonic annotations, an exhaustive search strategy would be prohibitively time-consuming in practice for any but the shortest musical extracts; one half measure can have over 100,000 valid derivations [8].

The **PVHM** uses *heuristic search* strategies to infer *latent structure* using the protovoice model, followed by *feature extraction* which finally allows the chord labels to be inferred using a *probabilistic model of harmony.*



(a) Modelling



(b) Inference

Figure 1.1: Problem overview

1.2 Related Work

Automatic chord estimation systems first emerged in in the 60's, making use of hand-crafted grammar/rule-based systems [27] [48], followed by the development of optimisation algorithms in the early 2000s [32]. In more recent years, supervised learning approaches have have risen in popularity, exploiting large datasets and improved compute power [31] [28] [25].

The protovoice model is the first to provide a unified theory that relates three aspects of tonal music analysis that are typically considered independently: voice-leading, how notes relate to each other *sequentially*; harmony, how notes relate to each other through *simulataneity*; and note function, how notes relate to each other through recursive *functional dependencies*. Previous models have been developed alongside parsing algorithms to perform automatic chord estimation that consider these dimensions of musical structure separately [27] [48], but in this project we use the relationship between these dimensions of music as the basis of heuristic design.

1.3 Achievements

This was an ambitious project; I met all the Success Criteria and completed the extension tasks. I show that the protovoice model can be used to effectively annotate pieces with chord labels, and these results provide a promising foundation for the model being developed further as a sophisticated tool for the automated analysis of western total music. The PVHM has been made **open source** to accommodate future research in the area.

Preparation

2.1 Probabilisitc Inference

2.1.1 Probabilistic Inference and Modelling

Bayesian inference is a statistical inference paradigm in which Bayes' theorem is used to update the probability for a hypothesis in light of available evidence. This allows us to *infer* information about *latent*(hidden) variables based on *observed* evidence.

The method of Probabilistic inference reduces to representing the *joint distribution* of all the random variables in a system, such that we can compute any probability of interest in that system. We typically Let X and Z denote the observed and latent(unobserved) variables respectively, and let Y denote the random variable we wish to infer. Then we are interested in representing the *joint distribution* $P(X, Y, Z)$, so that we can compute $\sum_Z P(Y|X, Z)$. **Modelling** is the process of approximating a joint distribution through abstract representations and relationships.

We first consider how we would compute the best hypothesis for Y given the evidence X, ignoring the latent variables Z.

There are two methods used to find the most likely hypothesis Y:

- The **maximum a priori estimate** maximises the *conditional likelihood* of the hypothesis given the evidence, given by $P(Y|X)$. This requires us to know the *prior* $P(Y)$, the distribution of the hypothesis, beforehand. The prior can be learned from labeled data.

$$\begin{aligned}\hat{Y} &= \arg\max_Y P(Y|X) \\ &= \arg\max_Y P(X|Y)P(Y)\end{aligned} \tag{2.1}$$

- The **maximum likelihood estimate** maximises the conditional likelihood of the observed evidence given the hypothesis, given by $P(X|Y)$. This method allows us to capture *uncertainty* in the prediction, given by $P(X|\hat{Y})$.

$$\hat{Y} = \arg\max_Y P(X|Y) \tag{2.2}$$

Definition 2.1.1 (Factoring)**.**

$$P(A, B) = P(B|A)\ P(A) \tag{2.3}$$

Definition 2.1.2 (Marginalisation)**.**

$$P(A) = \sum_B P(A|B)\ P(B) \tag{2.4}$$

Definition 2.1.3 (Chain Rule of Probability)**.**

$$P(A, B) = P(A|B)\ P(B) \tag{2.5}$$

2.1.2 Inferring Latent variables

To incorporate the latent variables Z, we use *marginalisation* and the *chain rule of probability* to show that:

$$\begin{aligned} P(Y|X) &= \sum_Z P(Y, Z|X) \\ &= \sum_Z P(Y|X, Z)\ P(Z|X) \end{aligned} \tag{2.6}$$

Given an abstract model $P(Z, X)$, we avoid summing over the values of Z by finding $\hat{Z} = \arg\max_Z P(Z|X)$, the most likely value of Z given X, using MLE or MAP estimation.

We thus update the prior $P(Z|X)$ as follows:

$$P(Z|X) = \begin{cases} 1 & \text{if } Z = \hat{Z} \\ 0 & \text{otherwise} \end{cases}$$

This gives us an approximation of $P(Y|X)$:

$$P(Y|X) \approx P(Y|X, \hat{Z}) \tag{2.7}$$

2.2 Overview of Approach

Probabilistic programming is a programming paradigm that makes use of model definitions and statistical inference algorithms to compute the conditional distribution of inputs that could have given rise to an observed output.

In the context of ACE, we consider the underlying sequence of chord labels $L_0, L_1, \ldots L_n$ as an **input**, and the musical surface or score as the **observed output** S.

In this sense, ACE can be solved by finding the most likely sequence of labels for the given surface, described by the equation:

$$\hat{L} = \arg\max_{L} P\left(L|S\right) \tag{2.8}$$

The difficulty arises from the complexity and prohibitively large number of the **latent variables $\boldsymbol{\phi}$**; in reality, we need to maximise $\sum_{\boldsymbol{\phi}} P(L|S, \boldsymbol{\phi})\ P(\boldsymbol{\phi}|S)$.

$$\hat{L} = \arg\max_{L} \sum_{\boldsymbol{\phi}} P(L|S, \boldsymbol{\phi})\ P(\boldsymbol{\phi}|S) \tag{2.9}$$

The set of latent variables $\boldsymbol{\phi}$ is **practically infinite**. These include the author's compositional conception, their musical conception, cognitive phenomena experienced by listeners, shared experience distilled into music theory, musical trends/culture and notational conventions.

This cannot be solved analytically, but we approximate $\hat{L}$ using models that encode domain specific knowledge about both the music generation and labelling processes, through joint conditional distributions.

Approximating Conditional Likelihood with MLE

Now consider the set of latent RVs $\boldsymbol{\phi}$ as the union of two *disjoint sets*, $\boldsymbol{\phi'}$ and $\boldsymbol{\phi''}$, where $\boldsymbol{\phi'}$ is the subset of $\boldsymbol{\phi}$ which we can *reasonably infer* given L and S, and $\boldsymbol{\phi''} = \boldsymbol{\phi} \setminus \boldsymbol{\phi'}$. Assuming $P(\boldsymbol{\phi'}|L)$ and $P(\boldsymbol{\phi''}|L)$ are independent, it follows:

$$\begin{aligned} P(L|S) &= \sum_{\boldsymbol{\phi'}} \sum_{\boldsymbol{\phi''}} P(L|S, \boldsymbol{\phi'}, \boldsymbol{\phi''})\ P(\boldsymbol{\phi'}, \boldsymbol{\phi''}|S) \\ &= \sum_{\boldsymbol{\phi'}} \sum_{\boldsymbol{\phi''}} P(L|S, \boldsymbol{\phi'}, \boldsymbol{\phi''})\ P(\boldsymbol{\phi'}|S)\ P(\boldsymbol{\phi''}|S) \end{aligned} \tag{2.10}$$

We take $P(\phi''|S)$ to be a uniform distribution as we have no prior knowledge of the latent variables that we cannot infer. Using the technique described in Equation 2.7, we find $\hat{\phi}'$, thus giving:

$$\hat{L} \approx \arg\max_{L} P(L|S, \hat{\phi}') \tag{2.11}$$

2.3 Inferring Structure

2.3.1 The Protovoice Model

The protovoice model is a formal generative model which represents a piece of music as a graph where each note is a node, and notes are connected by stepwise protovoice edges.

The Protovoice model is primarily concerned with the analysis of Western Classical music, although it could be applied to different musical styles, such as jazz or some popular western music [8].

2.3.2 Voices

The input we are concerned with is called a score, a symbolic abstraction of a piece of music based on a 2-dimensional axis.

The marks on on score represents notes, with the pitch of the note corresponding to its position on the vertical axis[1], and the notes' position in time represented by the horizontal axis.

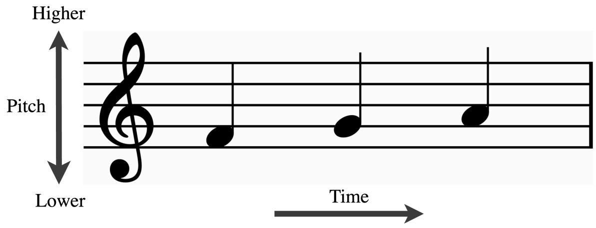


Figure 2.1: An example of music notation showing an ascending stepwise sequence.

The notion of a *voice* is crucial for the understanding of the protovoice model. A voice typically refers to a single melodic line (sequence of notes) that is part of a musical composition. The term is derived from its use in choral music, such as J.S Bach's four-voice chorales, which consist of 4 sung melodic lines. The term voice is used is used more generally however, the melodic lines do not need to be sung or voice-like in character and can be performed by any melodic instrument.

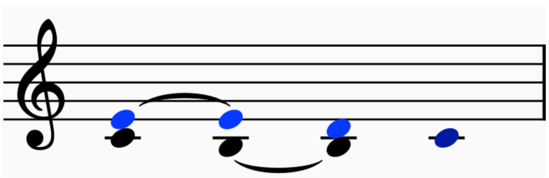


Figure 2.2: A short cadential phrase with two voices.

Polyphony refers to a piece of music that can contains more than one voice. Typically polyphonic music will have a set number of voices throughout the piece, but *free polyphony* refers to music where the number of voices is arbitrary and can change throughout the piece.

[1]This is a simplification as there are other factors that determine the pitch, such as the key signature, accidentals and intonation.

There are three types of relations between notes that form the basis for the protovoice model:

- **Horizontal**: As music is perceived in time, natural sequential relations arise between subsequent notes, in fact, we can define a total order on the notes of a piece of music based on their positions on the horizontal axis.
- **Vertical**: This refers to the pitch axis on the score. The vertical(pitch) arrangement of the notes determine the emergent harmony when they are perceived simultaneously. Typically, multiple *voices* heard together will lead to an emerging sequence of harmonies, which can be described using chord labels.

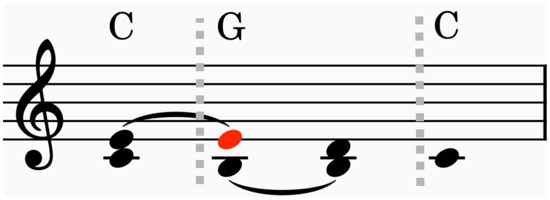


Figure 2.3: Chord labels shown with segment boundaries.

- **Functional**: Functional relations refer to the purpose or function of a note relative to another note. These functions can include repetitions, where both notes have the same *pitch*, or ornaments/neighbour notes, where the child note is a step (single unit of pitch) away from the parent note. These relations can be applied recursively, giving rise to a network of dependencies.

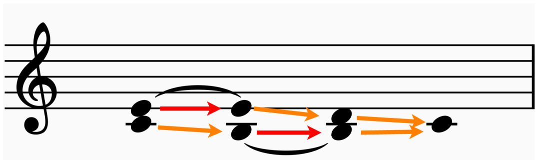


Figure 2.4: Functional dependencies between notes.

2.3.3 Inner Structure

The protovoice model generates a piece of music through sequential operations on notes, inserting new child notes by elaborating existing notes.

Protovoices are represented as a directed graph G with one vertex for each note, a vertex each for the beginning ($\rtimes$) and end ($\ltimes$) of the piece, and edges that indicate connections between notes. We define a note as $p \in \mathcal{P}$ where $\mathcal{P}$ is the set of pitches, i.e the set of vertical positions on the score[2]. A *protovoice* is a path within this graph. The protovoice model is characterised by stepwise generative operations on notes. A *repetition* inserts a child note of the same pitch on either side of the parent note, a *neighbour* inserts a child

[2]See Appendix C for a detailed explaination of the pitch representation used in this project

note that is a step away from the parent, and finally *passing* notes connect two protovoices stepwise.

These operations relate child notes to one or two *parent* notes. *Single-sided* operations are represented by attaching a new *child* note with an edge connected to a parent note:

$$p \implies x \to p \quad \text{or} \quad p \implies p \to x \tag{2.12}$$

Double-sided operations are represented by edge replacement.

$$p_1 \to p_2 \quad \implies \quad p_1 \to c \to p_2 \tag{2.13}$$

The generation of a piece begins with the empty piece ($\rtimes \to \ltimes$) and is defined by the recursive application of inner operations. The full set of rules is left to the appendix, but here are a few as an example:

$$\begin{aligned} x &\implies n \to x && \texttt{left-neighbor} \\ \rtimes \to \ltimes &\implies \rtimes \to x \to \ltimes && \texttt{root-note} \\ x \to y &\implies x \to x' \to y && \texttt{repeat-after'} \end{aligned} \tag{2.14}$$

2.3.4 Outer Structure

The inner structure provided by protovoices captures the sequential and functional organisation of notes, but does not capture when notes are simultaneous. To model simulataneity of notes we introduce *slices*, representing segments of a piece where a group of notes are heard, and *transitions* which contain the protovoice edges between notes in the two neighbouring slices. These provide a higher level abstraction that are used to capture more musical structure.

As slices and transitions contain notes and edges respectively, we call the slices and transitions *outer structure*, and the notes and edges contained therein *inner structure*.

Definition 2.3.1 (Multiset)**.** A *multiset* is a set that allows multiple instances for each of its elements, formally defined as an ordered pair (A, m) where A is the *underlying set* of the multiset, and $m : A \to \mathbb{Z}^+$ gives the *multiplicity*, such that the number of occurences of a in (A, m) is given by $m(a)$.

Definition 2.3.2 (Slice)**.** A *slice* $s \in \mathcal{S}$ is defined as a multiset of pitches $(\mathcal{P}, m)$.

$$s = \left\{p_1^{m(p_1)}, \ldots, p_n^{m(p_n)}\right\} \tag{2.15}$$

Definition 2.3.3 (Transition)**.** A *transition* $t \in \mathcal{T}$ relates two adjacent slices, s_l and s_r, with a configuration of edges e.

$$\mathrm{t} = (\mathrm{s}_l, e, s_r) \tag{2.16}$$

The slices and transitions form a graph given slices as nodes and transitions as edges (containing inner edges). However as transitions only relate sequentially adjacent slices, the outer structure is in fact a *path graph* and can thus be represented as a list of vertices.

Definition 2.3.4 (Path Graph). [3] A *path graph* is a graph which can be represented as an alternating sequence of elements from two sets A and B, defined inductively as:

$$\begin{aligned} P &= a & a &\in A \\ P &= abP & a &\in A, b \in B \end{aligned} \tag{2.17}$$

Definition 2.3.5 (Outer Structure).
The *outer structure* is thus defined as a path graph, represented as:

$$P = t_1\ s_1\ t_2\ s_2\ \dots\ t_n \qquad t_i \in \mathcal{T},\ s_i \in \mathcal{S},\ i \in 1 \dots n \tag{2.18}$$

The outer structure is generated by applying three operations described as production rules recursively:

- A **split** replaces a transition t by inserting a new slice s' and two surrounding transitions t_l and t_r. Each of the edges in t can have by one or more inner operations applied to it. The edges generated by these inner operations can either be discarded, or keep to form the new edges of t_l and t_r.

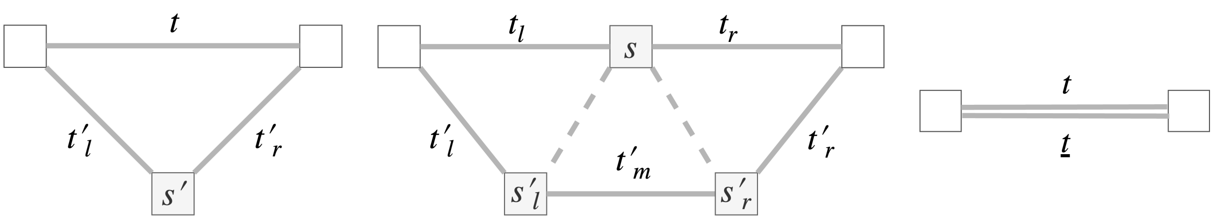
$$t \to t'_l\ s'\ t'_r \tag{2.19}$$

- A **spread** replaces a slice s by distributing its notes to two child slices s'_l and s'_r.

$$t_l\ s\ t_r \to t'_l\ s'_l\ t'_m\ s'_r\ t'_r \tag{2.20}$$

- A **freeze** marks a transition as terminal, such that the edges within the transition can no longer have operations applied to it.

$$t \to \underline{t} \tag{2.21}$$



(a) `split` (b) `spread` (c) `freeze`

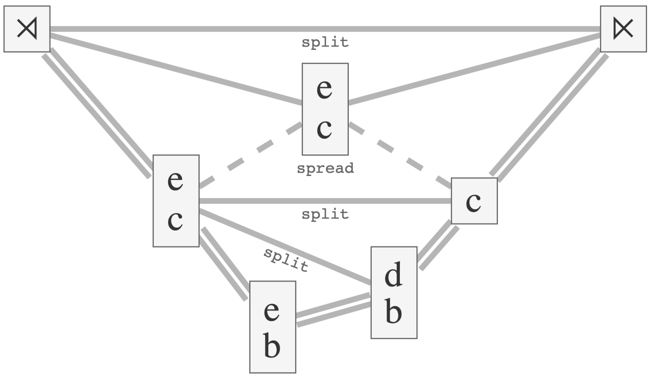
Figure 2.5: The three operations on outer structure.

Figure 2.5 shows a diagrammatic representation of each of these operations. The original slices and transitions are shown at the top of each diagram, while the lower part shows the generated structure after each outer operation is applied.

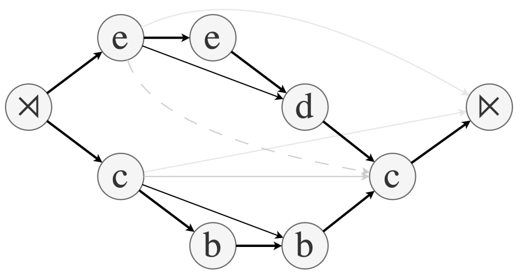
[3] This is refered to as a `Path` in the source code

The generation of a piece thus begins with the empty transition t_0, followed by a `split` operation, which generates the root slice of the piece.

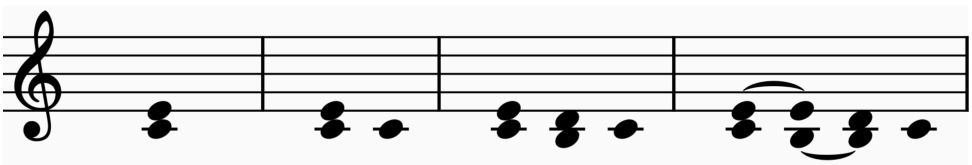
Figure 2.6 gives an example derivation of the short phrase shown in Figure 2.2. In Figure 2.6a, we can see that 4 outer operations have been applied to generate the surface. The spread operation introduces a passing edge between e and c, shown as the dashed line in Figure 2.6b. Note that the vertically aligned notes in Figure 2.6b correspond to the notes in the surface slices in Figure 2.6a, and the final surface shown on the right of Figure 2.6c.



(a) Outer structure



(b) Inner structure



(c) Abstract notation

Figure 2.6: An example derivation of the phrase shown in Figure 2.2.

2.4 Parsing

Shift-reduce parsers

Evaluating parses

Change all of this.

When parsing based on ambiguous grammars, we wish to find a *plausible derivation* of the given surface. For a derivation to be *plausible*, the *structure* it presents needs be consistent with a valid musical *interpretation*. Although this is non-trivial, plausibility can be approximately represented by a probability distribution, analogously to probabilistic context free grammars(PCFGs).

Definition 2.4.1 (Derivation plausibility)**.** The plausibility of a derivation is given by the product of the probabilities of each of the production rules used. Given a derivation D, its plausibility is defined:

$$p(D|S) = \prod_{D_i \in D} p(D_i|S) \tag{2.22}$$

Assuming we can calculate $p(d_i|S)$, we can find the most plausible derivation by taking the maximum likelihood estimate (MLE) of the distribution

$$\hat{D} = \arg\max_{D} p(D|S) \tag{2.23}$$

This presents **two key problems** to be solved:

- **Calculating** $p(d_i|\text{surface})$. Production probabilities can be viewed as parameters of the model; a common approach with PCFGs is to learn these parameters using machine learning. However, as the protovoice model is not context-free and the volume of data required is not available an alternative method must be found.
- **Combinatorial explosion**: Even if we could calculate $p(D|S)$ analytically, we would be prohibited by the large branching factor; a single piece can have upwards of $9^{9^{9^{\cdot^{\cdot^{\cdot^{9^9}}}}}}$ possible derivations[4].

2.5 Heuristic Search Algorithms

Heuristic search algorithms are introduced in 1B *Artificial Intelligence*, so I will therefore assume the reader has knowledge of the heuristic search paradigm.

The naive way to solve the above parsing problem is to use an *exhaustive search* strategy. This would theoretically allow us to find the most plausible derivation, but is computationally infeasible.

Best-first search is a heuristic search algorithm that selects the most *promising* node to expand based on a heuristic evaluation function. In general, the heuristic function $h(n)$ depends on the description of n, the description of the goal, information gathered by the search up to that point, and most importantly domain specific knowledge [33].

Beam search is an optimisation of best-first search that serves to reduce its memory requirements by only storing a limited number of best states as candidates to expand, dependent on the *beam width.* Beam search is greedy algorithm, so it does not necessarily produce the optimal solution, but trades optimality for improved complexity.

[4] I need a better way to represent the scale here. Perhaps I'll try to actually estimate a number

2.6 Inferring Labels

The task of inferring harmony (ACE) poses **three main challenges**:

- **Segmentation**: Segmentation is splitting of the score into segments which each have an individual label. For example, Figure 2.7 shows each segment separated by a dashed grey line. As these segments are typically not given a priori, both the segmentation and harmony needs to be inferred. Performing the joint task of segmentation and labelling is beyond the scope of this project, however, so we will assume that the segmentation is given.

- **Ambiguity**: The notes in a given segment may not be enough to determine the chord label. For example, the slice containing notes D and B in Figure 2.7 could be a realisation of a Bm triad, but the context of the neighbouring slices as well as the functional dependencies of the notes makes it clear that the label should be a G. Furthermore, ambiguity is often used with much license to create artistic interest.

- **Non chord-tones**: The slices in a given segment will typically consist of combination of *chord-tones* and *non chord-tones*. The chord tones directly define the harmony, so to perform ACE, an algorithm will need to distinguish between the two. Previous approaches have involved modelling generation as a noisy process, such that the non chord-tones are considered as noise [43]. The protovoice model allows us *explicitly* explain away non chord-tones. In Figure 2.7c, the non chord-tone is denoted in red. By applying a reduction that removes the neighbour note, E, we result in a single slice in that segment which only contains chord-tones(Figure 2.7d), thus describing the chord label more clearly, shown in Figure 2.7b.

[5]

2.7 Starting Point

2.7.1 Relevant courses and experience

Haskell I was introduced to Haskell during an internship during the summer before starting this project (July to August 2022). As a result, I had 2 months of experience with the language beforehand. I chose to use Haskell in order to further familiarise myself with the language, and because of its amenability to parsing algorithms. Furthermore, the functional paradigm and rich type system lends itself to modular software development.

[5] I'm considering adding a small section describing some relevant distributions. Dirchelet, Beta, Multinomial, categorical? It took a while for me to get my head around the fact that we used probability distributions of parameters of other probability distributions, e.g. sampling from Dirichlet for parameters of a multinomial or categorical.

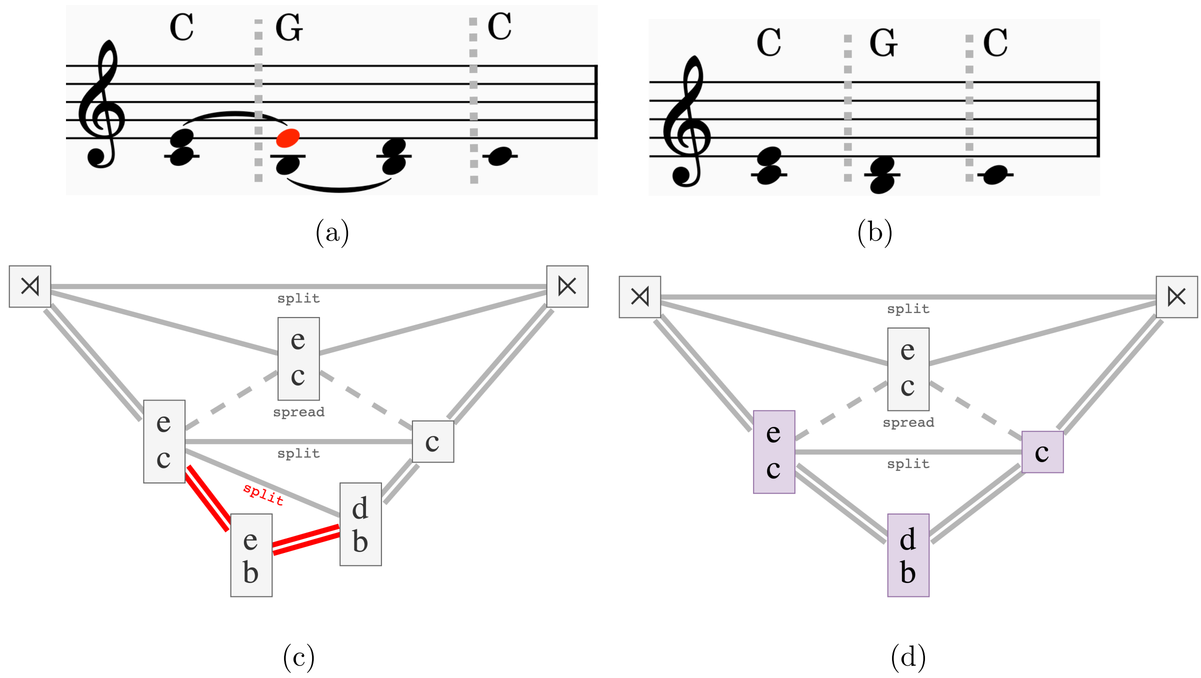


Figure 2.7: Applying a reduction to aid harmonic inference

Python I have experience coding in Python from personal projects as well as the 1A *Scientific Computing Practical Course*. This was chosen to made use of powerful existing libraries for handling large datasets and running experiments.

IB Two modules from 1B provide a foundation for this project. *Formal Models of Language* introduces the ideas and terminology used in the protovoice model, and *Artificial Intelligence* provides a background for classical search algorithms as well as some of the probabilistic frameworks used in the project.

2.7.2 Existing codebase

This project was built on a fork of the pre-existing *protovoices-haskell* GitHub repository. This repository contains custom data structures and types, allowing interoperability with other projects making use of the same model. I also make use of learned parameters from the implementation of the paper *Bayesian Model of Extended Chord Profiles* [9], as learning these parameters would be beyond the scope of this project.

2.8 Requirements Analysis

Success Criteria

The Success Criteria are given in the Project Proposal. During the preparation phase, I refined the Success Criteria by gaining clarity from the literature related to the project.

This project will be deemed a success given it achieves:

-

Risk Assessment

Table 2.1 shows a list of project deliverables with associated priorities and risk, denoted qualitatively. There is a high general risk in this project due to the fact that the protovoice model is a result of an unpublished thesis, and thus may have flaws that have not been discovered. The highest risk task is designing and implementing the protovoice parser. The heuristic design task is expansive, requiring creativity, research and iterative development, hence the risk is high. The baseline inference deliverable comprises a standard method for ACE, so there is minimal risk.

Table 2.1: Project Deliverables

ID	Deliverable	Priority	Risk
`core1`	Harmony Model	High	Low
`core2`	End to End Pipeline	High	Medium
`core3`	Protovoice Parser	High	High
`base1`	Baseline Reduction	High	Low
`base2`	Random Search	High	Low
`ext1`	Heuristic Design	Medium	High
`ext2`	Greedy Search	Medium	Medium
`ext3`	Beam Search	Low	Very High
`ext4`	Dual Beam Search	Low	Very High

2.9 Software Engineering Techniques

2.9.1 Development model

Based on the risk analysis (Table 2.1), I created a plan of which modules to implement in which order, and a list of milestones on a 2 week basis. I used Notion to maintain a list of core tasks and corresponding subtasks with associated priorities, which I used when deciding what to work on. My development strategy drew from the Agile methodology, working on tasks in two-week long sprints, with regular re-evaluations of the plan informed by experimental data and testing.

I made use of GitHub's continuous integration features to run a test suite on the repository after every commit.

2.9.2 Languages, libraries and tools

Table 2.2 shows a justified list of the key languages, libraries and tools used in the project.

Table 2.2: Languages, libraries and tools

Tool	Purpose	Justification	License
Languages			
Haskell	Main language used for the core, baseline and extension implementations	Protovoice model implementation is in Haskell. Functional and amenable to parser development.	GHCL
Python	Secondary language for experiments and analysis	Powerful library ecosystem for running experiments and creating plots	PSFL
Libraries			
Musicology	Haskell Library with data-types for pitches	Contains a robust implementation of spelled pitch classes, which would be tedious to reimplement.	BSD-3.0
Timeit	Lighweight wrapper to show the used CPU time of a monadic computation	This is used to time the runtime of the algorithms as part of analysis	BSD-3.0
Dimcat	Python library: DIgital Musicology Corpus Analysis Toolkit	This library was written to work with the datasets used in this project	GPL-3.0
Numpy	Python library used for preprocessing and analysis	Powerful standard library that is used in conjunction with Seaborn to run analysis and visualise data	BSD-3.0
Pandas	Python library for preprocessing and analysis	This is a standard library for data manipulation and processing	BSD-3.0
Seaborn	Python data visualisation library used for analysis	Creates high quality graphs and charts	BSD-3.0
Tools			
Docker	Containerised software service used to run repeatable experiments	The use of many libraries creates a web of dependencies to be resolved. Ensures the code will last and can be executed on different devices	Free/Paid
Git	Version Control, Continuous Integration	Provides natural backups and allows for reverts to previous commits if necessary	GPL-3.0
GitHub	Hosting source code	Free, reliable hosting	GPL-3.0
GHC	Compiling and profiling.	This is the standard Haskell compiler.	BSD-3.0
Stack	Haskell building and testing	Creates reliable builds, and includes a powerful testing framework.	BSD-3.0
Undotree	Vim Plugin: stores all past actions as a tree	Solves the problem of linear undo history being lost. Protects code between commits.	BSD-3.0
MuseScore	Music notation software	The raw inputs are in the MusicXML format, which is used by MuseScore 3	GPL-3.0
PAT	Protovoice Annotation Tool, Used to view protovoice derivations on a web browser	The protovoice derivations are huge and very complex, so it's vital to have a viewing tool for use in analysis and iterative development	GPL-3.0

2.9.3 Licensing

As shown in Table 2.2, I determined and read up on all of the licensing agreements for the tools used in the project. For the most part, these are all permissive licenses, guaranteeing freedom to use, modify and redistribute as well as permitting proprietary derivative works.

2.9.4 Datasets

2.9.5 Hardware, version control and backup

The code was developed using Vim for Haskell and Visual Studio Code for Python notebook development, on my personal laptop (16' MacBook Pro 2022, M1 Max, 32GB). Experiments were run first locally, then on HPC provided by the EPFL Digital Cognitive Musicology Lab (Dell PowerEdge R740XD Server, 2x Xeon Gold 625R, 768GB), using Jupyter notebooks to conduct the evaluation. I used GitHub for all my notes, development and dissertation writing. Finally, this dissertation was written in Vim with VimTeX.

Implementation

3.1 ProtoVoice Harmony Model

3.1.1 Overview

We present a **novel** inference system, the **ProtoVoice Harmony Model** that approximates $\arg\max_L P(L|S)$ to infer harmony from free polyphony.

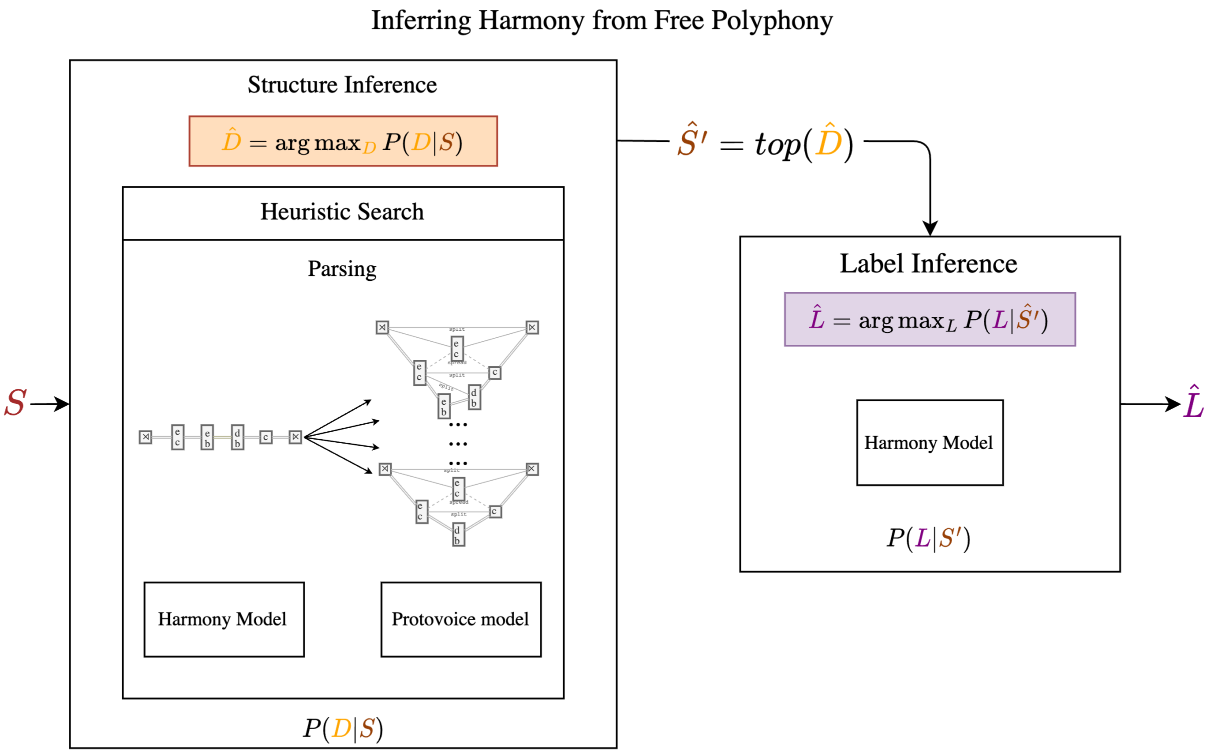


Figure 3.1: Overview of the inference process.

The PVHM is described by the following equation:

$$\hat{L} = \arg\max_L \Big(P(S|L, \hat{S}', \hat{D}) \cdot P(L) \Big) \tag{3.1}$$

Where $\hat{D}$ denotes the inferred *protovoice derivation* of the surface, obtained using a modified beam search algorithm that explores the state space of valid partial derivations of the S, guided by a heuristic $\mathcal{H}_{\prime}$. $\hat{S}$ is the associated reduction of the surface, given by $top(\hat{D})$.

The core of this model is that we find the most likely reduction $\mathcal{R}(S)$ of the surface S by inferring the most likely protovoice structure $\hat{D}$. The baseline algorithm achieves through a random search through the space of valid partial derivations. The extension algorithms use an informed heuristic H_0 to guess $P(D|S)$ in order to guide the search, and explore the trade-offs between different search algorithms. Given the inferred derivation we extract the corresponding reduced surface with $top : \mathbf{D} \to \mathbf{S}'$. Given the reduced surface $\mathcal{R}(S)$ we finally solve $\hat{L} = \arg\max_L P(L|S, \mathcal{R}(S)) \cdot P(L)$, finding the most likely sequence of chord labels $\hat{L}$ given the reduced surface $\mathcal{R}(S)$.

Development plan

This system is modular by design, leading to a natural progression of complexity throughout the implementation; the baseline models provide a subset of the functionality required for the full system. For example, the harmony model is a prerequisite for the first baseline, but is also used in all subsequent models. [1]

- The Baseline Reduction computes a simplified $\mathcal{R}(S)$ using random sampling.
- The Random Search uses $\mathcal{R}(S) = top(D_0)$, where D_0 is the derivation found by randomly choosing each parse operation.
- The first extension involves developing an informed heuristic $\mathcal{H}_0$
- The Greedy Search uses heuristic $\mathcal{H}_0$ to approximate $\arg\max_D P(D|S)$.
- The Beam Search uses $\mathcal{H}_0$ to implement a search parameterised by the beam width β
- The Dual Beam Search implements a custom search algorithm using $\mathcal{H}_0$, parameterised by α and β, which control operation dependent beam widths.

3.2 Repository Overview:

Repository Justification

The repository has been split into four main folders, with the addition of `Main.hs` which serves as an interface between the python experiment code and the algorithms developed in Haskell.

[1] This could be complemented by a diagram showing the progression

- Firstly, the `src/Core/` folder contains all the core code, including the implementation of the parsing and inference functions using the probabilistic model of harmony, as well as some helper code for file handling.
- The `experiments/` folder contains all the python code that is used for this project. The experiments consist of three stages, as described by the three main files: `preprocess.py`, `experiments.py` and `analysis.py`. Splitting these stages up prevents wasteful computation, as all the pre-processing can be done just once, while experiments are run on the processed data iteratively alongside algorithm development.
- The `src/Algorithms/` folder contains all the parsing algorithms including the baseline and extension search algorithms. Having all the algorithms contained in one module allows experiments to be run using any selection of algorithms and input data, facilitating the evaluation process.
- Finally, the `test/` folder contains unit tests and end-to-end tests for use in Continuous Integration.

Figure 3.2 illustrates how these modules are connected.

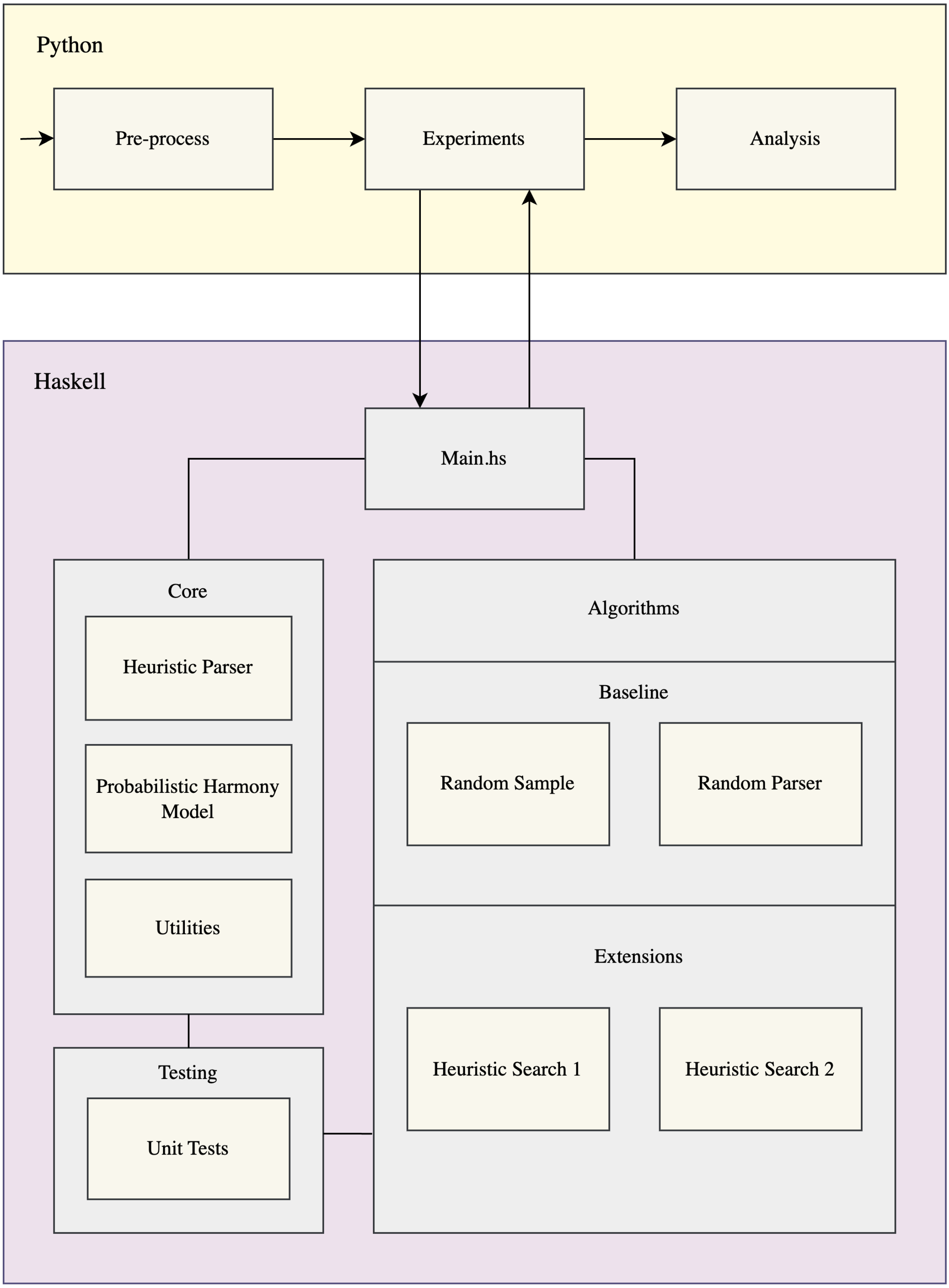


Figure 3.2: Block diagram of project components

Table 3.1: Repository Overview

File/Folder	Description	LOC
`protovoices-haskell/`	Root directory	2272
`├── src/`		
`│   ├── HeuristicParser.hs, HeuristicSearch.hs`	Core Implementation (Section x)	470
`│   ├── RandomChoiceSearch.hs, RandomSampleParser.hs`	Baseline Implemetation (Section x)	121
`│   ├── Heuristics.hs, PBHModel.hs`	Extension Implementation (Section x)	383
`│   ├── FileHandling.hs`	Utilities	188
`│   └── ...`		
`├── app/`		
`│   └── MainFullParse.hs`	Entry Point	431
`├── harmonic-inference`		
`├── experiments/`	Running Experiments	115
`│   ├── preprocess.ipynb`		
`│   ├── experiments.ipynb`		
`│   ├── analysis.ipynb`		
`│   ├── dcml_params.json`		
`│   └── inputs/`		
`└── test/`	Unit Tests (Section x)	611

3.3 Core Implementation

3.3.1 End to End Pipeline

The first step was to implement a full end-to-end pipeline from piece to chord label predictions. This was achieved by writing `preprocess.py`, `runExperiment.py` and `analysis.ipynb`. The repository was then containerised using *Docker*, so that experiments could be run and analysed using a DCML server with automated dependency resolution. A *jupyter-lab* environment was set up to run the full pipeline, allowing analysis to be conducted through a browser. Experiments were executed on the server, initialised via *ssh*, and `tmux` was used to maintain the server environment across connections, with automated scripts to *fetch* and *pull* the latest changes from GitHub.

Dataset selection

The datasets used contain a selection or corpuses, bodies of work by a single composer as shown below, these were selected to exhibit a range of different styles. Each dataset contains an entire set work by a composer consisting of between $15 - 50$ annotated scores, each with varying lengths.

We divide each dataset into training, validation, and test sets, using a 60:20:20 split, using *stratified sampling* in order to maintain a balanced representation of the different composers.

Choosing Hyperparameters

In order to choose hyperparameters, we combine the training and validation sets from each of the five datasets into a single training pool, then use 5-fold cross-validation using the training pool.

Final evaluation plan

For the final evaluation, the performance of all developed algorithms will be evaluated on the hold-out test set (20%). This is to provide an unbiased estimate of each algorithm's performance on **unseen** data.

We are also assessing the trade-off between accuracy and performance.

3.3.2 Harmony Model

The first step was to implement the harmony model that allows us to compute $P(L|S')$ and $P(L|S)$. This provides a framework to evaluate all developed algorithms. We use log probabilities.

Predicting labels from reduced slices

We have a set of pitches $\mathcal{P}$ and a set chord-types $\mathcal{C}$. A chord label is defined as a tuple of *root note* pitch and chord-type: $\mathcal{L} = (\mathcal{P}, \mathcal{C})$. Consider a slice $s = \left\{p_1^{m(p_1)}, \ldots, p_n^{m(p_n)}\right\}$ where $p_i \in \mathcal{P}$. We consider all possible chord labels $l \in \mathcal{L}$.

The goal is to find:

$$\hat{l} = \arg\max_{l} P(l|s) \tag{3.2}$$

We choose to find $P(l|x)$ rather than $P(s|l)$ as it gives us a *confidence* in our prediction, given by $P(\hat{l}|s)$.

We consider case that all pitches in the slice s are *chord-tones* of the actual label. This is because there is a direct relationship between the chord-tones present in a segment, and the corresponding label.

We model the generation of a slice s of size n from a label l as follows:

$$\begin{gathered} s \sim \text{Multinomial}(n, |\mathcal{P}|, \boldsymbol{\Phi}_l) \\ \text{where } \boldsymbol{\phi}_l^{(p)} = P(\text{Pitch} = p | \text{Label} = l) \end{gathered} \tag{3.3}$$

We define the slice vector $\boldsymbol{v}(s) = [\ m(p_1)\ m(p_2)\ \ldots\ m(p_k)\ ]$ where $k = |\mathcal{P}|$. Then $P(l|s)$ is given by: [2]

$$\begin{gathered} \begin{aligned} P(l|s) &= P(s|l) \cdot P(l) \\ &= f(\boldsymbol{v}(s), \boldsymbol{\phi}_l) \cdot P(l) \end{aligned} \\ \text{where } f \text{ is the multinomial probability density function} \\ f(x_1, \ldots, x_k; p_1, \ldots, p_k) = \frac{\Gamma\left(\sum_i x_i + 1\right)}{\prod_i \Gamma(x_i + 1)} \prod_{i=1}^{k} p_i^{x_i} \end{gathered} \tag{3.4}$$

[2]Struggling with naming here. Using P for pitches conflicts the P for probability.

The log probability $\log P(l|s)$ is given by

$$\begin{aligned} \log P(l|s) &= \log P(s|l) + \log P(l) \\ &= \cdots \\ &= \log\left(\Gamma\left(\sum_i x_i + 1\right)\right) + \sum \left(x_i \log p_i^{x_i} - \log \Gamma(x_i + 1)\right) + \log P(l) \end{aligned} \tag{3.5}$$

So to find the most likely label l given slice s we take:

$$\hat{l} = \arg\max_l \left(\log\left(\Gamma\left(\sum_i x_i + 1\right)\right) + \sum \left(x_i \log p_i^{x_i} - \log \Gamma(x_i + 1)\right) + \log P(l)\right) \tag{3.6}$$

The full derivation has been omitted for the sake of brevity. Given that $|\mathcal{P}|$ is finite (30? I forgot) we can solve this directly.

3.3.3 Protovoice Parser

The Protovoice Parser is a bottom-up parser for the protovoice model that enumerates all the valid protovoice derivations that produce a given surface: $S \to [D]$. A derivation is described as a sequence of rule applications in *left-most derivation order*, applied to a fully reduced surface, S'.

Formally, a derivation D is defined as a pair (top, ops), where the $surface$, S, is derived by starting with the fully reduced *top* and applying each operation in *ops* in order.

It is informative to consider the *generation order* and *parse order* of a protovoice derivation. In generation order, we begin with the reduced surface $S' = top(D)$ with a pointer at the left-most transition and apply each `split` or `spread` operation to the two left-most non-terminal transitions in the path graph, generating new slices and transitions. The `freeze` operation marks a transition as terminal thus shifts the pointer to the right. Operations are applied until the pointer is right-most, and all transitions are terminal, resulting in the fully elaborated surface S and a derivation $D = (top, ops)$, where *ops* is the sequence of operations applied.

Parsing is the inverse of generation: the parse begins with the elaborated surface S consisting of only frozen transitions with a pointer at the right-most frozen transition. During the parse, we can either `unfreeze` the transition at the pointer, shifting the pointer to the left, or apply a `unsplit` or `unspread` reduction to the two transitions to the right of the pointer. Once the pointer is left-most, all transitions are unfrozen and there is a *single reduced slice* for each chord segment at the top of the derivation, the resulting derivation is $D = (top, ops)$, where the reduced surface is $S' = top$.

This is a form of *shift-reduce* parser, making one pass across the surface, right to left. The state of the parse is represented by `parseState` and implements `expand: parseState` $\to$ `[parseState]` such that we can use search algorithms to conduct the parse.

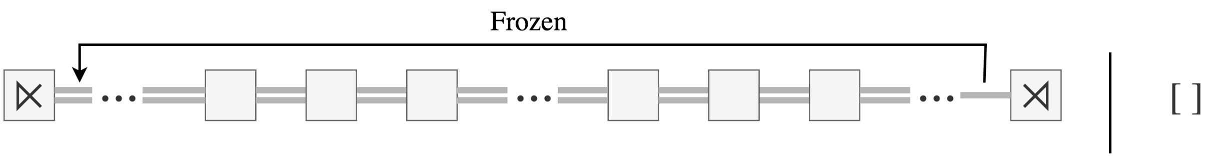


Figure 3.3: `pSFrozen`

In the initial parse state, `pSFrozen`, the surface is represented as a `path` from the end to the beginning of the piece, with the right-most transition at its head. The start($\rtimes$) and stop($\ltimes$) symbols are not explicitly stored in the path, but are implied.

All transitions are initialised as frozen, indicated by the double lines in Figure 3.3. The `path` begins on the right. In this initial state the only option is to unfreeze the rightmost transition, moving to the `pSSemiOpen` state.

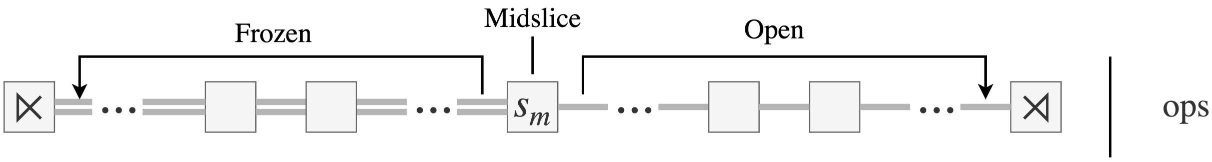


Figure 3.4: `pSSemiOpen`

The `pSSemiOpen` represents the majority of the parse. The pointer is represented by `midSlice`, and points to the rightmost frozen transition. The `open` path contains all unfrozen transitions (denoted by a single line) from the right of the pointer to the end of the piece, and the `frozen` path contains all frozen transitions from the pointer to the start of the piece.

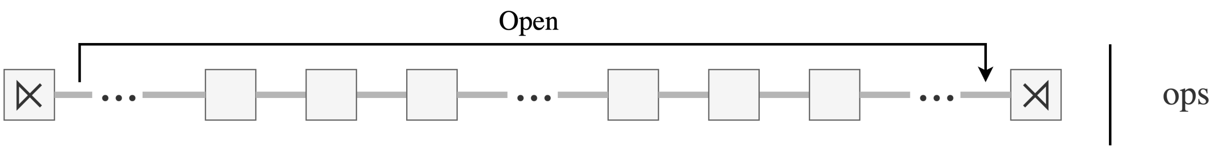


Figure 3.5: `pSOpen`

In the final state, `pSOpen`, the pointer is at the beginning of the piece, and comprises a single path `open` containing only unfrozen transitions from the beginning to the end of the piece.

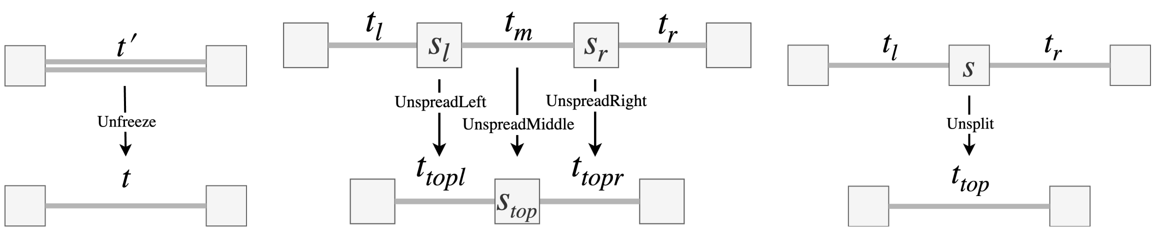
Each state also stores a list of operations, *ops* that when applied to the current reduction, results in the original surface. This is empty at the beginning of the parse; for each reduction applied, the corresponding generative operation *op* is consed to the list: $ops' := op : ops$.

This is not the only way to parse according to the protovoice model; the protovoice reduction rules can be applied to any pair of open transitions. Allowing reductions at any point along the surface could allow for different parse strategies to be employed that may have advantages. I choose not to pursue this due to the associated combinatorial explosion, and the benefits of being able to represent a protovoice derivation as a sequence of left-most reductions.

Parsing Operations

The `protoVoiceEvaluator` [10] provides methods that are used to enumerate the possible operations for each reduction type [3]. It also includes an implementation of a greedy parser, from which ideas were adapted and expanded upon to create the protovoice parser. This allows the parser to consider only the outer structure of slices and transitions while parsing without exposing the internal notes and edges.

$$\begin{aligned}
\texttt{evalUnfreeze}: &\quad \mathrm{t} \rightarrow [(t', op)] \\
\texttt{evalUnsplit}: &\quad (\mathrm{t}_l, s_l, t_m) \rightarrow [(t_{top}, op)] \\
\texttt{unSpreadLeft}: &\quad (\mathrm{s}_l, t_l) \rightarrow s_{top} \rightarrow [t_{topl}] \\
\texttt{unSpreadRight}: &\quad (\mathrm{s}_r, t_r) \rightarrow s_{top} \rightarrow [t_{topr}] \\
\texttt{unSpreadMiddle}: &\quad (\mathrm{s}_l, t_{top}, s_r) \rightarrow \texttt{Maybe}\ (s_{top}, op)
\end{aligned} \tag{3.7}$$



(a) `Unfreeze` (b) `Unspread` (c) `Unsplit`

Figure 3.6: Reduction operations.

Boundary handling

Recall that the goal of the parse is to reduce the original surface such that there is one slice per segment. In order to achieve this, constraints are imposed on the reduction operations dependent on adjacent segment boundaries.

Each transition has a boolean *boundary* value which indicates if the transition is across a segment boundary, denoted by the dashed vertical line in Figure 3.7. Let $B_t : t \rightarrow \{\mathbf{True}, \mathbf{False}\}$, such that B_f, B_l, B_m, and B_r denote the boundary values of t_f, t_l, t_m and t_r respectively. As an `unspread` operation ($t_l\ s_l\ t_m\ s_r\ t_r \rightarrow t_{topl} s_{top} t_{topr}$) merges two slices s_l and s_r, removing t_m, the constraint $\neg B_m$ is imposed to prevent two segments from merging. Similarly, an `unsplit` operation (eg. $t_l\ s_l\ t_m \rightarrow t_{top}$) combines two transitions,

[3] Note that the notation used here for functions combines type definitions with variable names. For example, the function `evalUnsplit` has type `evalUnsplit` :: $(transition, slice, transition) \rightarrow [(transition, operation)]$, but type names (e.g. *transition*) have been substituted by variable names (e.g. t_l, t_r, t_{top}) for expository purposes.

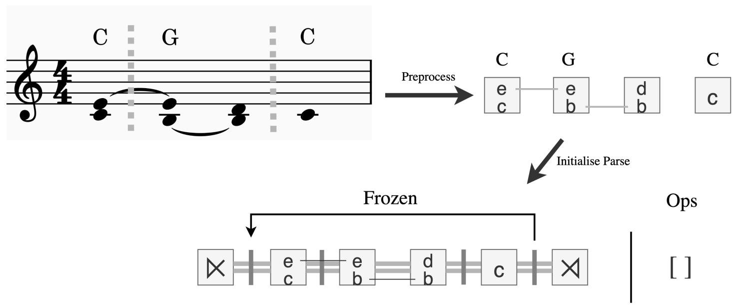


Figure 3.7: Parse Initialisation

thus we impose the constraint $\neg(B_l \wedge B_m)$. Finally, the `unfreeze` operation shifts the context to the left. If a boundary transition is unfrozen, no more reduction operations can be applied to its right, hence it a necessary condition that the segment has been fully reduced. Thus the constraint imposed on an `unfreeze` operation is $\neg B_f \vee (B_l \wedge B_m \wedge B_r)$. Note that there are some configurations that are *unreachable*, such as $B_f \wedge (B_l \wedge B_m \wedge \neg B_r)$, thus the `unfreeze` constraint could be simplified to $\neg B_f \vee (B_l \wedge B_m)$, but I have chosen not to as it obfuscates the semantics, and the cost of an additional logic check is negligible. [4]

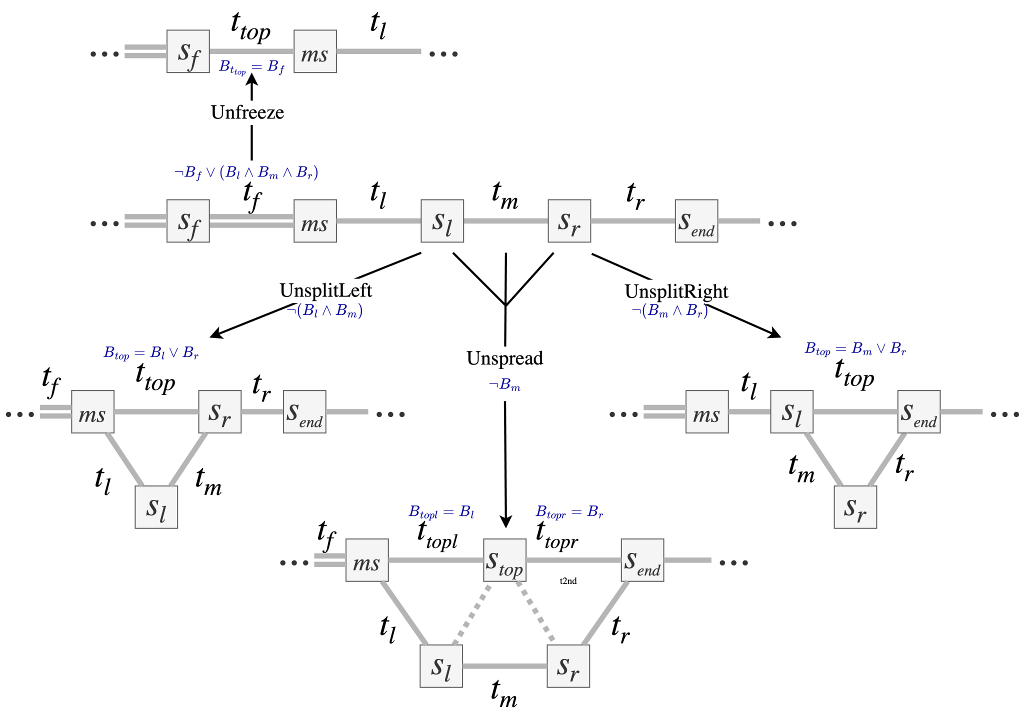


Figure 3.8: Boundary reduction conditions and propagation

Figure 3.8 provides an illustration of these constraints. The boundary must also be propagated to parent transitions following a reduction, defined by the union of each child transitions' boundary values for a given reduction.

[4] Should this be a footnote? I have chosen to include it as it an example of a decision made that demonstrates thinking.

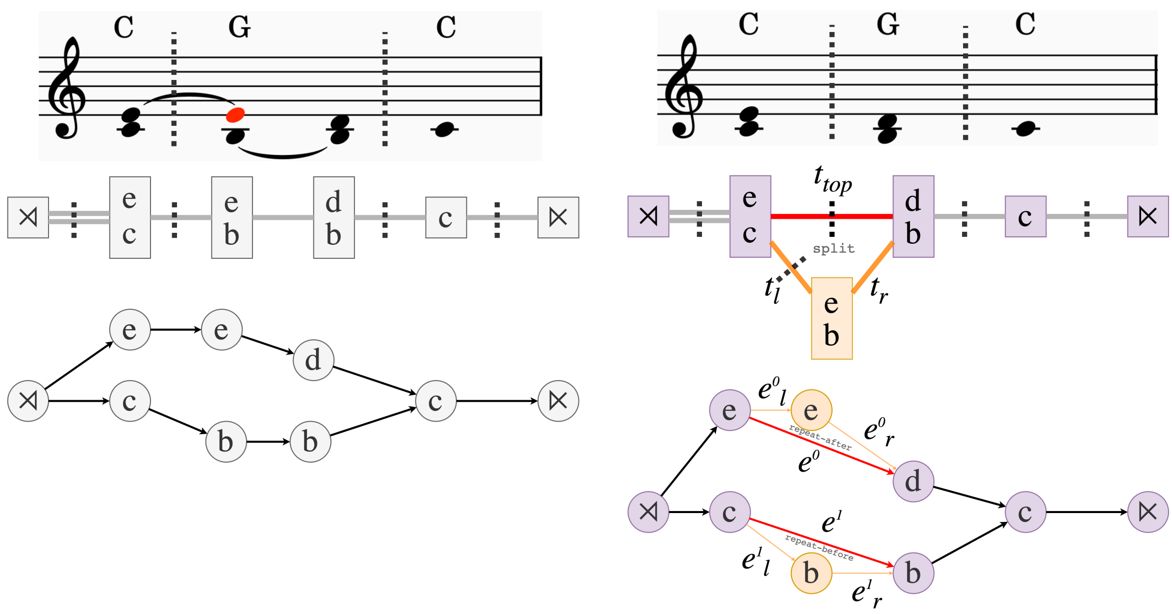
The `evalUnfreeze` and `evalUnsplit` functions allow the associated generative operation, `freeze` and `split` to be found trivially, but the possible `spread` operations need to derived using `unSpreadLeft`, `unSpreadRight`, and `unSpreadMiddle`. This is achieved using the *list monad*, in which multiple branches of sequential computation appear to be executed simultaneously, returning the results of all possible computation branches in a list.

Algorithm 1 Enumerate unspread reductions

function COLLECTUNSPREADS(s_l t_l s_m t_r s_r)
  using *ListMonad* **do**
  $(s_{top}, op) \leftarrow$ `maybeToList $ evalUnspreadMiddle`(s_l, t_m, s_r)
  $t_{topl} \leftarrow$ `evalUnspreadLeft` (s_l, t_l) s_{top}
  $t_{topr} \leftarrow$ `evalUnspreadRight` (s_r, t_r) s_{top}
  return $(l_{top}, s_{top}, r_{top})$
end function

Finally, given that an `unspread` reduction involves two transitions, and an `unsplit` reduction involves only one, it is important to ensure the parse pointer is such that there are two open transitions to its right wherever possible. This avoids the case of applying alternating `unsplit` and `unfreeze` reductions, as there is only zero or one open transition within the current segment, and `unspread` reductions require two. This is not important for the sake of a parse, which enumerates all derivations, but is important later on when designing heuristic search algorithms. [5]

Parse Steps



(a) Before (b) After

Figure 3.9: Single Reduction Step

[5] I guess I'll move this to the heuristic design section?

Figure 3.10 shows the possible outer structures for a single segment reduction consisting of three slices and four transitions. This corresponds to the `pSOpen` case as there are no frozen transitions. Note that while the number of possible outer stucture reductions is 3^n given n slices, each reduction consists of a set of inner operations which also grows exponentially with the size of each slice.

The parse states define a **directed acyclic graph** where the parseStates are nodes and the parse operations are edges. Traversing this graph results in either reach a valid derivation or dead state with no edges. It is possible to get stuck due to the constraint that protovoices are strictly stepwise, so there not be any possible inner operations to apply to further reduce the surface. This is rare in practice, so when running the search we allow it to restart if it reaches a dead state, bounded by a constant k.

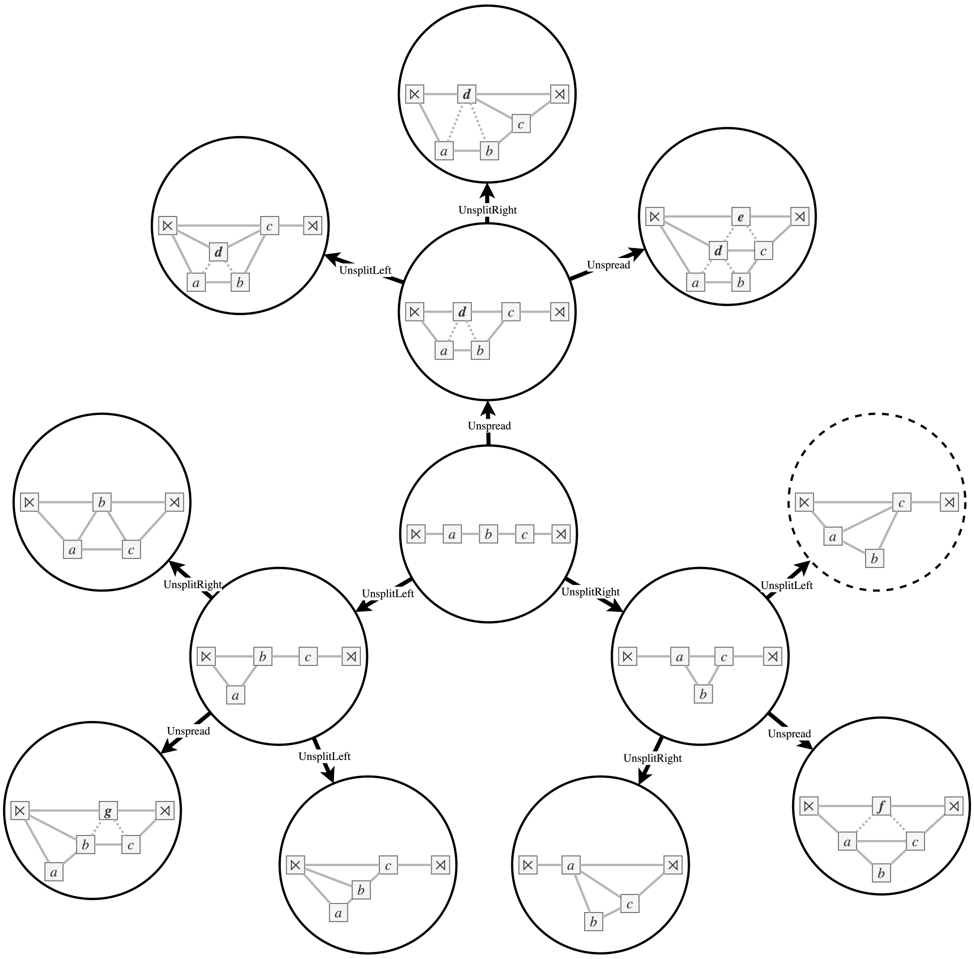


Figure 3.10: Parse state transitions for an open segment with three slices.

3.3.4 Baseline algorithms

The goal of protovoice parser is to find a derivation D which results in a top surface $S' = top(D)$, consisting of only chord-tones.

The simplest possible baseline is one that generates slices randomly for each segment, without considering the piece at all. This is the **RandomSample** algorithm. While this is naive, it provides a useful reference for evaluation metrics.

As a second baseline, consider the simplest form of reduction, removing notes randomly until there is a reasonable number left. This is the **RandomReduction** algorithm. It is equivalent to combine all the slices in a given segment and take a random sample of n notes from the combined slice. The number of notes is sampled from a Poisson distribution, $n \sim \text{Poisson}(\lambda)$, where λ is learned from data.

Next, we implement the **RandomWalk** algorithm using the protovoice parser. This algorithm takes a random walk through the parse states graph. The protovoice model does not make judgements based on harmony or function explicitly, but by comparing this algorithm to the other baselines, we can explore whether the constraints enforced by using a protovoice derivation has any effect on the quality of the reduction. We show that this works surprisingly well.

The `ParseAlgo` type-class provides an interface for running these algorithms. The result is wrapped in a `Maybe` as it is possible for the search to get stuck, and is in the `IO` monad to allow non-determinism.

```
class (Show algo) => ParseAlgo algo where
  runParse :: algo -> AlgoInput -> IO (Maybe AlgoResult)
```

Listing 3.1: Algorithm type-class

3.4 Extension Implementation

3.4.1 Parsing with Heuristics

Let us take a step back and consider what has achieved up to this point. Recall our goal is to find $\arg\max_L P(L|S)$.

We are able to find a protovoice derivation of the piece, D and corresponding reduced surface S', which allows us to use the harmony model to infer the labels.

This is described by $\arg\max_L \sum\limits_{D,S'} (P(S|D)P(D|S')P(S'|L))P(L)$. It is not possible to enumerate D in the general case, so we approximate this by finding $\hat{D}$ and $\hat{S}'$. There is an abstract relation between derivations D and surfaces S as a given derivation can correspond to many different surfaces and vice-versa. This abstract relation is described by $P(D,S) = P(D) \cdot P(S|D)$.

The probability of the derivation P(D) is defined $\prod_i P(d_i|d_0,\ldots,d_{i-1})$ where d_0 is the first step in the generative direction. The plausibility of the derivation from the generative direction is thus defined $P(D|S) = P(d_i|d_{>i}, S)$. This cannot be calculated, but we take an approximation that is akin to markov assumption, evaluating a single operation.

We have shown that we can use the protovoice model to reduce the surface. The baseline model achieves this by applying reductions randomly, so that all the notes in original surface are derived through recursive elaborations of the reduced surface, *top*().

The protovoice model does explicitly make any judgements based on the vertical or functional relations between notes. The idea of the heuristic design is to incorporate domain specific knowledge relating to these relations, and use this to guide the parse.

The likelihood of each derivation step is approximated by considering only local information: the top surface and the reduction operation applied.

We need to create a function $\mathcal{H} : \texttt{(ParseState, Op)} \rightarrow \texttt{Cost}$.

3.4.2 Extending the Harmony Model

Predicting chordness

This is to guide the heuristic search.

We define *chordness* as the likelihood that a given slice could have been generated by the most likely label $\hat{l}$.

$$\text{Chordness}(s) = P(s|\hat{l})P(\hat{l}|s) \tag{3.8}$$

where $\hat{l} = \arg\max_l P(l|s)$ as above.

Ornamentation likelihood

Pitches that are not chord-tones are referred to as *ornaments*.

3.4.3 Heuristic Design

In order to calculate the cost of a single reduction, we consider the plausibility of the corresponding operation being applied in the generative direction. For example, if during the parse we are in state S_i and apply a `unsplit` operation, to which takes us to S_j, we calculate the plausibility of the generative `split` operation in state S_j. Recall that the protovoice model comprises an inner and outer structure. The outer structure provides an abstraction of the inner structure, allowing the parsing algorithm to be designed using operations applied to slices and transitions.

Recall that the core of the protovoice model is a set of generative operations on notes.

Any operation consists of a set of regular edges, which are either neighbour

Step 1: Design heuristic to be as accurate as possible. I.e the extreme is to consider every possible parse, but for a single piece there can be over $10^{10^{10^{10^{10^{10}}}}}$ different parses. We consider 1 step at a time at first - this still results in needing to choose an operation out of upwards of 30,000,000 options for just a single step.

First the full piece heuristic parse

The goal of the heuristic is to find the probability of a reduction.

Here the `ProbVectorSingle` function a vector for the.

Algorithm 2 Heuristic function $\mathcal{H}_0$

```
function EvaluateOperation(State, Op)
    if Op is a Split then
        return EvaluateSplit(State, Op)
    else if Op is a Spread then
        return EvaluateSpread(State, Op)
    else if Op is a Freeze then
        return Log(1)                                          ▷ Likelihood of 1
    end if
end function

function EvaluateSpread(State, Op)
    s_l, s_r ← ParentSlices(State, Op)
    E ← {RegularEdges(Op) ∪ PassingEdges(Op)}
    LogLikelihoods ← {EvaluateEdge(e, s_l, s_r) : e ∈ E}
    return Average(LogLikelihoods)
end function

function EvaluateSplit(State, Op)                ▷ How much detail? keepEdges?
    s_l, s_r ← ParentSlices(State, Op)
    e_reg ← RegularEdges(Op)
    e_pass ← PassingEdges(Op)
    e_single ← LeftEdges(Op) ∪ RightEdges(Op)
    e_right ← RegularEdges(Op)
    LogLikelihoods ← {EvaluateEdge(e, s_l, s_r) : e ∈ {e_reg ∪ e_pass}}
    return Average(LogLikelihoods)
end function

function EvaluateEdge(e, s_l, s_r)
    n ← child note introduced in e
    if e elaborates two parents, p_l and p_r then
        φ⃗ ← ProbVectorDouble(n, p_l, p_r, s_l, s_r)
    else if e elaborates single left parent p then
        φ⃗ ← ProbVectorSingle(n, p_l, s_l)
    else if e elaborates a right parent p then
        φ⃗ ← ProbVectorSingle(n, p_r, s_r)
    end if
    return CategoricalLogPdf(φ⃗, n)
end function
```

Scoring Unsplit Operations

Consider the Split rule :

$$t \rightarrow t'_l \; s' \; t'_r$$

During a split, each edge in the transition and each node in an adjacent slice can be elaborated by one or more inner operations. These new edges can be discarded or kept to form the new edge of t'_l and t'_r.

The notes in the child slice s can either have edges connected to the left neighboring slice or right neighbouring slice, or both. In other words each note in the child slice was generated as an elaboration of the left or right parent, or both in the case of a double sided operation.

The plausibility of fa

previous note, subsequent note, both, or repetition of prev note, subsequent note etc. So we consider the chord tone profiles of the involved slices.

We first guess the chord type each parent slice.

$$\theta_l = \underset{c \in C}{argmax}\, P(s_l|c) \quad , \quad \theta_r = \underset{c \in C}{argmax}\, P(s_r|c)'$$

We now consider each edge individually, considering their likelihoods based on the proabilistic model of harmony along with theoretical assumptions.

Single Sided Operations

- Right Neighbour (Left Neighbour anagolously)

$$x \implies x \rightarrow n \quad , x, n \in P$$

$$x \sim \text{Categorical}(\sigma_{ct}^{\theta_l})$$

$$n \sim \text{Categorical}(\sigma_{or}^{\theta_r})$$

  Find

$$P(x, n \mid \theta_l)$$

- Right Repeat (Left Repeat anagolously)

$$x \implies x \rightarrow x \quad , x \in P$$

$$x \sim \text{Categorical}(\sigma_{ct}^{\theta_l})$$

  Find

$$P(x \mid \theta_l)$$

Two Sided Operations

- Root Note: This operation is only done once in the original model. In our case we do not need to consider due to segment boundaries.
- Full Repeat:

$$x \implies x \to n \quad , x, n \in P$$

$$x \sim \text{Categorical}(\sigma_{ct}^{\theta_l})$$

$$n \sim \text{Categorical}(\sigma_{or}^{\theta_r})$$

  Find

$$P(x, n \mid \theta_l)$$

- Left Repeat of Right:

$$x \to y \implies x \to y' \to y$$

$$y \sim \text{Categorical}(\sigma_{ct}^{\theta_l})$$

  Find

$$P(y \mid \theta_l)$$

- Full Neighbour:

$$x_1 \to x_2 \implies x_1 \to n \to x_2, x \in P$$

  Find

$$P(\mid \theta_l, \theta_r)$$

Scoring Unspread Operations

Consider the Spread rule :

$$t_l s_r \to t_l' s_l t_m' s_r t_r'$$

We make the assumption that s, s_l, & s_r are all realisations of the same chord. This lines up with the music theorretical basis for this operation in the model(justify).

Thus we find the most likely chord (optional extension: marginalise over all chords)

$$\theta = \underset{c \in C}{argmax}\, P(s|c)$$

When then measure the extent to which the parent slics match this chord.

$$p(s_l, s_r|\theta)$$

We can calculate $p(s_l|\theta)$ and $p(s_r|\theta)$ using the multinomial distribution probability density function as described in the preparation chapter.

Scoring Unfreeze Operations

We assign 0 cost to unfreeze operations. This means we need to be careful about ensure that we don't just unfreeze the entire piece immediately. Careful construction of the search algorithm can ensure this. More later.

Full state evaluation

Figure 3.11: What does this do?

function FindMysteries(S, k)
    if $k \leq 1$ **then**
        return $\{\}$ ▷ The empty set $\emptyset$
    end if
    $n \leftarrow |S|$ ▷ Cardinality of S
    $k \leftarrow \textsc{Min}(k, n+1)$ ▷ Since maximum of n mysteries
    $r \leftarrow \lfloor \lfloor k/2 \rfloor (\frac{n+1}{k}) \rfloor$ ▷ 1-indexed rank of the pivot
    $p \leftarrow \textsc{MedianOfMediansSelect}(S, r)$ ▷ element of rank r
    $S_1 \leftarrow \{x \in S : x < p\}$
    $S_2 \leftarrow \{x \in S : x > p\}$
    return $\textsc{FindMysteries}(S_1, \lfloor k/2 \rfloor) \cup \{p\} \cup \textsc{FindMysteries}(S_2, \lceil k/2 \rceil)$
end function

We need to combine all of these in a fair way. Also the distinction between splits and spreads need to be considred, as they are different operations, the calculations of likelihood may cause an imbalance. All likelihoods are stored in log space.

3.4.4 Heuristic Search

Given the heuristic $\mathcal{H}_0$, the first option is to use a greedy search. We keep unfreeze operations as an option.

We find that the number of operations causes a combinatorial blowup, so the search doesn't end.

Problem of very large slices.
Segment by segment heuristic parse - avoids the problem, but is slightly hacky. Can we incorprate our knowledge regarding the relative proportion of chord tones and ornaments. Should we allow duplicates of notes in slices? Perhaps we should favour spreads more.

Always consider a certain number of slices and spreads.

Algorithm 3 Greedy Search

initialise: $state \leftarrow initialState$
while $state$ is not a goal state **do**
$state \leftarrow \operatorname*{arg\,min}_{s \in \text{EXPAND}(state)} \mathrm{H}(s)$
$freezes, spreads, splits \leftarrow$ Split $nextStates$ by $operation$ type
$open \leftarrow$ Take best β best states from $freezes \cup spreads \cup splits$
end while
return Best state in open

3.4.5 Beam Search

Step 2: Relax the heuristic search in order to reduce runtime/ lower complexity.

In the case that there are 85,000,000 options, perhaps we should sample the options rather than evaluating all of them.

This version of heuristic search should be able to parse full pieces (hopefully), so can be used to compare with the baselines on an entire corpus.

Beam of size n, with 1 for a freeze, k for spread, n-k-1

3.4.6 Dual Beam Search

It isn't clear how to determine how we should balance the costs of unspread and unsplit op

Algorithm 4 MultiBeam Search

hyper-parameters: $\beta \leftarrow BeamWidth, \gamma \leftarrow ReservoirSize$
initialise: $open \leftarrow (initialState, 0)$
while $open$ does not contain any goal states **do**
$nextStates \leftarrow \bigcup_{(s,c) \in open} \{(s', c' + \mathrm{H}(s')) : (s', c') \in \text{EXPAND}((s,c))\}$
$freezes, spreads, splits \leftarrow$ Split $nextStates$ by $operation$ type
$open \leftarrow$ Take best β best states from $freezes \cup spreads \cup splits$
end while
return Best state in open

3.4.7 Stochastic Dual Beam Search

We propose sampling from the options, increasing the proportion that we ignore dependent on the number of options.

Here H assigns a cost for moving from state s to s'. This is defined as the negated log likelihood in $\mathcal{H}$.

Algorithm 5 Reservoir MultiBeam Search

hyper-parameters: $\beta \leftarrow BeamWidth, \gamma \leftarrow ReservoirSize$
initialise: $open \leftarrow (initialState, 0)$
while *open* does not contain any goal states **do**
  $nextStates \leftarrow \bigcup_{(s,c) \in open} \{(s', c' + \textsc{h}(s')) : (s', c') \in \textsc{Expand}((s, c))\}$
  $freezes, spreads, splits \leftarrow$ Split *nextStates* by *operation* type
  $unFreezes \leftarrow \bigcup_{s \in freezes} (\textsc{ReservoirSample}(\gamma, s))$
  $unSpreads \leftarrow \bigcup_{s \in spreads} (\textsc{ReservoirSample}(\gamma, s))$
  $unSplits \leftarrow \bigcup_{s \in splits} (\textsc{ReservoirSample}(\gamma, s))$
  $open \leftarrow$ Take best β best states from $unFreezes \cup unSpreads \cup unSplits$
end while
return Best state in open

3.5 Choosing hyper parameters

3.6 Testing

3.6.1 Unit Tests

3.6.2 Qualitative Tests

Throughout the design of the heuristic H_0 and the implementation of different search algorithms, a few segments we used as reccuring test examples.

Evaluation

In this chapter, I provide qualitative and quantitative evaluations of the work completed. I then provide and interpret evidence to show that the success criteria were met.

The main questions to answer are as follows:

- *Can the proto-voice model be used to accurately infer chord labels?*
- *Can the proto-voice model be used to practically infer chord labels?*
- *How well my heuristic search algorithms infer chord labels?*

4.1 Metrics

Ambiguity. which metrics exist? discussion. Accuracy and likelihood.

4.2 Harmony Model

4.2.1 Accuracy

When evaluating using the protovoice model: we assume that we result in only chord tones for each segment. Thus we use the chord tone probabilities to evaluate the prediction.

When just using a random sample, we have to assume that there is a mixture model of chord tones and ornaments. We use the learnt parameters to determine the distribution.

These two measures of likelihoods are comparable as they are drawn from the same distributions.

We also need to infer chord labels. We can simply choose the chord that is most likely according to our model.

This gives us two key metrics, likelihood and accuracy.

Could also use a more sophisticated notion of accuracy, using a chord similarity function [16]. The `mir_eval` package provides a plethora of metrics to compare chord label predictions [37].

4.2.2 Sensitivity Analysis

4.2.3 Qualitative Analysis

4.3 Baseline Algorithms

Reminder of what the search is actually doing.

Things to note

- The fact that segmentation is known ahead of time provides a great deal of information [13]

- So we use comparisons between the random sample from each segment algorithm and the random parse algorithm to see if the use of the grammar provides an advantage over just sampling the notes directly, without looking at relations between notes.

- Then we want a heuristic search algorithm that considers each option exhaustively and finds the best local option. This is too computationally expensive to be used for whole pieces.

- Given there can be millions of possible next states in the search, we need to look at different strategies to avoid searching through them all. E.g just sample states.

- Sensitivity Analysis for the heuristic search is useful for the evaluation. Explore how robust it is to handcrafted attacks/ different types of passages.

- Could evaluate by segments instead of pieces.

4.4 Extension Algorithms

4.4.1 Accuracy

4.4.2 Scalability

4.4.3 Qualitative Analysis

4.5 Success Criteria

I've shown that i met success criteria x by this analysiss. etc.

4.6 Limitations

I'm solving:

$$\hat{L} = \arg\max_{L} \left(p(S|L)\right) \tag{4.1}$$

But I could be solving:

$$\hat{L} = \arg\max_{L} \left(p(S|L)p(L)\right) \tag{4.2}$$

In which case:

To compute the conditional probability $p(L|S)$, we use Bayes' theorem:

$$p(L|S) = \frac{p(S|L)\ p(L)}{P(S)} \tag{4.3}$$

Finding the most likely sequence of labels is found using:

$$\arg\max_{L} \left(\underbrace{p(S|L)}_{\text{likelihood}}\ \underbrace{p(L)}_{\text{prior}}\right) \tag{4.4}$$

The prior probability of a chord sequence $p(L)$ can be learned from a labeled dataset of chord sequences, and the likelihood can be found using a probabilistic harmony model. The likelihood $p(S|L)$ can be found using

This would be better, but was beyond the scope of the project.

Conclusions

In this chapter, I first discuss the success achieved by the project then offer a reflection on lessons learned. Finally, I consider the directions in which there is potential for future work.

5.1 Achievements

5.2 Lessons learned

5.3 Future Work

Bibliography

[1] Chen, T.-P., and Su, L. Functional Harmony Recognition of Symbolic Music Data with Multi-task Recurrent Neural Networks, Sept. 2018.

[2] Chen, T.-P., and Su, L. Harmony Transformer: Incorporating Chord Segmentation into Harmony Recognition. In *International Society for Music Information Retrieval Conference* (2019).

[3] Cohen, D. E. "The Imperfect Seeks Its Perfection": Harmonic Progression, Directed Motion, and Aristotelian Physics. *Music Theory Spectrum 23*, 2 (Oct. 2001), 139–169.

[4] Davidson-Pilon, C. Bayesian Methods for Hackers, Mar. 2023.

[5] Deng, J., and Kwok, Y.-K. Large vocabulary automatic chord estimation using bidirectional long short-term memory recurrent neural network with even chance training. *Journal of New Music Research 47*, 1 (Jan. 2018), 53–67.

[6] Ebcioğlu, K. An Expert System for Harmonizing Four-Part Chorales. *Computer Music Journal 12*, 3 (1988), 43–51.

[7] Feisthauer, L., Bigo, L., Giraud, M., and Levé, F. Estimating keys and modulations in musical pieces. In *Sound and Music Computing Conference (SMC 2020)* (June 2020).

[8] Finkensiep, C. *The Structure of Free Polyphony.* PhD thesis, EPFL, Lausanne, 2023.

[9] Finkensiep, C., Ericson, P., Klassmann, S., and Rohrmeier, M. Chord Types and Ornamentation - A Bayesian Model of Extended Chord Profiles. *Open Research Europe* (Apr. 2023).

[10] Finkensiep, C., and Rohrmeier, M. Modeling and Inferring Proto-voice Structure in Free Polyphony. In *Proceedings of the 22nd ISMIR Conference* (Online, Nov. 2021).

[11] Gjerdingen, R. Cognitive Foundations of Musical Pitch Carol L. Krumhansl. *Music Perception: An Interdisciplinary Journal 9* (July 1992), 476–492.

[12] Goodman, J. Semiring Parsing. *Computational Linguistics 25*, 4 (1999), 573–606.

[13] Gotham, M., Kleinertz, R., Weiss, C., Müller, M., and Klauk, S. What if the 'When' Implies the 'What'?: Human harmonic analysis datasets clarify the relative role of the separate steps in automatic tonal analysis. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR 2021, Online, November 7-12, 2021* (2021), J. H. Lee, A. Lerch, Z. Duan, J. Nam, P. Rao, P. van Kranenburg, and A. Srinivasamurthy, Eds., pp. 229–236.

[14] Granroth-Wilding, M. Harmonic Analysis of Music Using Combinatory Categorial Grammar. *The University Of Edinburgh* (2013).

[15] Harasim, D. Harmonic Syntax in Time: Rhythm Improves Grammatical Models of Harmony. In *Proceedings of the 20th ISMIR Conference* (2019), T. J. O'Donnell and M. A. Rohrmeier, Eds., ISMIR.

[16] Humphrey, E. J., and Bello, J. P. Four Timely Insights on Automatic Chord Estimation.

[17] Johannes, H. Ms3 - Parsing MuseScore 3. https://github.com/johentsch/ms3, 2021.

[18] Kidney, D. O., and Wu, N. Algebras for weighted search. *Proceedings of the ACM on Programming Languages 5*, ICFP (Aug. 2021), 1–30.

[19] Koops, H. V., de Haas, W. B., Bransen, J., and Volk, A. Automatic chord label personalization through deep learning of shared harmonic interval profiles. *Neural Computing and Applications 32*, 4 (Feb. 2020), 929–939.

[20] Koops, H. V., de Haas, W. B., Burgoyne, J. A., Bransen, J., Kent-Muller, A., and Volk, A. Annotator subjectivity in harmony annotations of popular music. *Journal of New Music Research 48*, 3 (May 2019), 232–252.

[21] Korzeniowski, F., and Widmer, G. Improved Chord Recognition by Combining Duration and Harmonic Language Models, Aug. 2018.

[22] Krumhansl, C. L., and Kessler, E. J. Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys. *Psychological Review 89* (1982), 334–368.

[23] Lerdahl, F., and Jackendoff, R. *A Generative Theory of Tonal Music*, repr. ed. MIT Press, Cambridge, Mass., 2010.

[24] Marsden, A. Schenkerian Analysis by Computer: A Proof of Concept. *Journal of New Music Research 39*, 3 (Sept. 2010), 269–289.

[25] Masada, K., and Bunescu, R. Chord Recognition in Symbolic Music: A Segmental CRF Model, Segment-Level Features, and Comparative Evaluations on Classical and Popular Music, Oct. 2018.

[26] Mauch, M., Mullensiefen, D., Dixon, S., and Wiggins, G. Can Statistical Language Models be used for the Analysis of Harmonic Progressions?

[27] Maxwell, H. J. An expert system for harmonizing analysis of tonal music. In *Understanding Music with AI: Perspectives on Music Cognition.* MIT Press, Cambridge, MA, USA, Aug. 1992, pp. 334–353.

[28] McLeod, A., and Rohrmeier, M. A Modular System for the Harmonic Analysis of Musical Scores using a Large Vocabulary. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR 2021, Online, November 7-12, 2021* (2021), J. H. Lee, A. Lerch, Z. Duan, J. Nam, P. Rao, P. van Kranenburg, and A. Srinivasamurthy, Eds., pp. 435–442.

[29] Mearns, L. The Computational Analysis of Harmony in Western Art Music.

[30] Neuwirth, M., Harasim, D., Moss, F. C., and Rohrmeier, M. The Annotated Beethoven Corpus (ABC): A Dataset of Harmonic Analyses of All Beethoven String Quartets. *Frontiers in Digital Humanities 5* (July 2018), 16.

[31] Ni, Y., Mcvicar, M., Santos-Rodriguez, R., and De Bie, T. An end-to-end machine learning system for harmonic analysis of music, July 2011.

[32] Pardo, B., and Birmingham, W. P. Algorithms for Chordal Analysis. *Computer Music Journal 26*, 2 (June 2002), 27–49.

[33] Pearl, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* The Addison-Wesley Series in Artificial Intelligence. Addison-Wesley Pub. Co, Reading, Mass, 1984.

[34] Pickens, J. *Harmony Modeling for Polyphonic Music Retrieval.* PhD thesis, University of Massachusetts, 2004.

[35] Quinn, I. Are Pitch-Class Profiles Really "Key for Key"? *Zeitschrift der Gesellschaft für Musiktheorie [Journal of the German-Speaking Society of Music Theory] 7* (Jan. 2010).

[36] Radicioni, D. P., and Esposito, R. BREVE: An HMPerceptron-Based Chord Recognition System. In *Advances in Music Information Retrieval*, J. Kacprzyk, Z. W. Raś, and A. A. Wieczorkowska, Eds., vol. 274. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 143–164.

[37] Raffel, C., McFee, B., Humphrey, E. J., Salamon, J., Nieto, O., Liang, D., and Ellis, D. P. W. Mir_eval: A Transparent Implementation of Common MIR Metrics. In *Proceedings of the 15th International Conference on Music Information Retrieval* (2014).

[38] Raphael, C., and Stoddard, J. Functional Harmonic Analysis Using Probabilistic Models. *Computer Music Journal 28*, 3 (Sept. 2004), 45–52.

[39] Rocher, T., Robine, M., Hanna, P., and Strandh, R. Dynamic Chord Analysis for Symbolic Music.

[40] Sapp, C. Visual hierarchical key analysis. *Computers in Entertainment 3* (Oct. 2005), 1–19.

[41] Sapp, C. 6 Computational Chord-Root Identification in Symbolic Musical Data: Rationale, Methods, and Applications.

[42] Sidorov, K., Jones, A., and Marshall, D. MUSIC ANALYSIS AS A SMALLEST GRAMMAR PROBLEM.

[43] Temperley, D. An Algorithm for Harmonic Analysis. *Music Perception 15*, 1 (Oct. 1997), 31–68.

[44] Temperley, D. A Bayesian Approach to Key-Finding. In *Music and Artificial Intelligence*, G. Goos, J. Hartmanis, J. van Leeuwen, C. Anagnostopoulou, M. Ferrand, and A. Smaill, Eds., vol. 2445. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 195–206.

[45] Temperley, D. A Bayesian Approach to Key-Finding. In *Music and Artificial Intelligence* (Berlin, Heidelberg, 2002), C. Anagnostopoulou, M. Ferrand, and A. Smaill, Eds., Lecture Notes in Computer Science, Springer, pp. 195–206.

[46] Temperley, D. A Unified Probabilistic Model for Polyphonic Music Analysis. *Journal of New Music Research 38*, 1 (Mar. 2009), 3–18.

[47] Volk, A. Improving Audio Chord Estimation by Alignment and Integration of Crowd-Sourced Symbolic Music. 141–155.

[48] Winograd, T. Linguistics and the Computer Analysis of Tonal Harmony. *Journal of Music Theory 12*, 1 (1968), 2–49.

Additional Information

Project Proposal

Inferring Harmony from Free Polyphony

Judah Daniels

April 19, 2023

DOS: Prof. Larry Paulson

Crsid: jasd6
College: Clare College

B.1 Abstract

A piece of music can be described using a sequence of chords, representing a higher level harmonic structure of a piece. There is a small, finite set of chord types, but each chord can be realised on the musical surface in a practically infinite number of ways. Given a score, we wish to infer the underlying chord types.

The paper *Modeling and Inferring Proto-voice Structure in Free Polyphony* describes a generative model that encodes the recursive and hierarchical dependencies between notes, giving rise to a grammar-like hierarchical system [10]. This proto-voice model can be used to reduce a piece into a hierarchical structure which encodes an understanding of the tonal/harmonic relations of a piece.

Christoph Finkensiep suggests in his paper that the proto-voice model may be an effective way to infer higher level latent entities, such as harmonies or voice leading schemata. Thus in this project I will ask the question: is this parsing model an effective way to annotate harmonies? By 'effective' we are referring to two things:

- Accuracy: can the model successfully emulate how experts annotate harmonic progressions in musical passages?
- Practicality: can the model be used to do this within a reasonable time frame?

While the original model could in theory be used to generate harmonic annotations, its exhaustive search strategy would be prohibitively time-consuming in practice for any but the shortest musical extracts; one half measure can have over 100,000 valid derivations [10]. My approach will be to explore the use of heuristic search algorithms to solve this problem.

B.2 Substance and Structure

B.2.1 Core: Search

The core of this project is essentially a search problem characterised as follows:

- The state space S is the set of all possible partial reductions of a piece along with each reduction step that has been done so far.
- We have an initial state $s_o \in S$, which is the empty reduction, corresponding to the unreduced surface of the piece. The score is represented as a sequence of slices grouping notes that sound simultaneously. We are also given the segmentation of the original chord labels that we wish to retrieve.
- We have a set of actions, A modelled by a function $action : A \times S \to S$. These actions correspond to a single reduction step.

  - The reduction steps are the inverses of the operations defined by the generative proto-voice model.

- Finally we have a goal test, $goal : S \to \{true, false\}$ which is true iff the partial reduction s has exactly one slice per segment of the input.

  - This means the partial reduction s contains a sequence of slices which start and end positions corresponding to the segmentation of the piece.

- At the first stage, this will be implemented using a random graph search algorithm, picking each action randomly, according to precomputed distributions.

B.2.2 Core: Evaluation

The second core task is to create an evaluation module that iterates over the test dataset, and evaluates the partial reduction computed by the search algorithm above. This will be done by comparing the outputs to ground truth annotations from the Annotated Beethoven Corpus.

In order to do this I will make use of the statistical harmony model from Finkensiep's thesis, *The Structure of Free Polyphony* [8]. This model provides a way of mapping between the slices that the algorithm generates and the chords in the ground truth. This can be used to empirically measure how closely the slices match the expert annotations.

B.2.3 Extension

Once the base search implementation and evaluation module have been completed, the search problem will be tackled by heuristic search methods, with different heuristics to be trialled and evaluated against each other. The heuristics will make use of the chord profiles from Finkensiep's statistical harmony model discussed above. These profiles relate note choices to the underlying harmony. Hence the heuristics may include:

- How the chord types relate to the pitches used.

- How the chord types relate which notes are used as ornamentation, and the degree of ornamentation.

- Contextual information about neighboring slices

B.2.4 Overview

The main work packages are as follows:

Preliminary Reading – Familiarise myself with the proto-voice model, and read up on similar models and their implementations. Study heuristic search algorithms.

Dataset Preparation – Pre-process the Annotated Beethoven Corpus into a suitable representation for my algorithm.

Basic Search – Implement a basic random search algorithm that takes in surface and segmentations, and outputting the sequence of slices matching the segmentations.

Evaluation Module – Implement an evaluation module to evaluate the output from the search algorithm.

End-to-end pipeline – Implement a full pipeline from the data to the evaluation that can be used to compare different reductions.

Heuristic Design – Extension – Trial different heuristics and evaluate their performance against each other.

Dissertation – I intend to work on the dissertation throughout the duration of the project. I will then focus on completing and polishing the project upon completion.

B.3 Starting Point

The following describes existing code and languages that will be used for this project:

Haskell – I will be using Haskell for this project as it is used in the proto-voice implementation. It must be noted that my experience with Haskell is limited, as I was first introduced to it via an internship this summer (July to August 2022).

Python – Python will be used for data handling. I have experience coding in Python.

Prior Research - Over the summer I have been reading the literature on computational models of music, as well as various parsing algorithms such as semi-ring parsing [12], and the CYK algorithm, which is used in the implementation of the proto-voice model.

Protovoices-Haskell – The paper *Modeling and Inferring Proto-Voice Structure in Free Polyphony* [10] includes an implementation of the proto-voice model in Haskell. A fork of this repository will form the basis of my project. This repository includes as parsing module which will be used to perform the actions in the search space of partial reductions. There is module that can exhaustively enumerate reductions of a piece, but this is infeasible in practice due to the blowup of the derivation forest.

MS3 – This is a library for parsing MuseScore Files and manipulating labels [17], which I will use as part of the data processing pipeline.

ABC – The *Annotated Beethoven Corpus* [30] contains analyses of all Beethoven string quartets composed between 1800 and 1826), encoded in a human and machine readable format. This will be used as a dataset for this project.

B.4 Success Criteria

This project will be deemed a success if I complete the following tasks:

- Develop a baseline search algorithm that uses the proto-voice model to output a partial reduction of a piece of music up to the chord labels.

- Create an evaluation module that can take the output of the search algorithm and quantitatively evaluate its accuracy against the ground truth annotations by providing a score based on a statistical harmony model.

- Extension: Develop one or more search algorithms that use additional heuristics to inform the search, and compare the accuracy with the baseline algorithm.

B.5 Timetable

Time frame	Work	Evidence
Michaelmas (Oct 4 to Dec 2)		
Oct 14 to Oct 24	*Oct 14*: Final proposal deadline. Preparation work: familiarise myself with the dataset and the proto-voice model implementation. Work on manipulating reductions using the proto-voice parser provided by the paper.	None
Oct 24 to Nov 7	Dataset preparation and handling.	Plot useful metrics about the dataset using Haskell
Nov 7 to Nov 21	Random Search implementation	None
Nov 21 to Dec 5	Evaluation Module. Continue with search implementation.	Evaluate a manually created derivation and plot results
Vacation (Dec 3 to Jan 16)		
Dec 5 to Dec 11	Evaluate performance of random search. Begin to work on extensions	Plot results
Dec 10 to Dec 21	Trial different heuristics. Implement an end-to-end pipeline from input to evaluation.	None
Dec 21 to Dec 27	None	None
Dec 27 to Jan 10	Continue trialing and evaluating heuristics	*Fulfill success criterion: At least one heuristic technique gives better performance than random search.*
Lent (Jan 17 to Mar 17)		
Jan 4 to Jan 20	Buffer Period to help keep on track	None
Jan 20 to Feb 3	*Feb 3*: Progress Report Deadline. Write progress report and prepare presentation. Write draft *Evaluation* chapter	Progress Report (approx. 1 page)
Feb 3 to Feb 17	Prepare presentation.	*Feb 8 – 15*: Progress Report presentation
Feb 17 to Mar 3	*Feb 17*: How to write a Dissertation briefing. Write draft Introduction and Preparation chapters. Incorporate feedback on Evaluation chapter.	Send draft Introduction and Preparation chapter to supervisor
Mar 3 to Mar 17	Write draft Implementation chapters. In-	Send draft Implemen-

B.6 Resources

I plan to use my own laptop for development: MacBook Pro 16-inch, M1 Max, 32GB Ram, 1TB SSD, 24-core GPU.

All code will be stored on a GitHub repository, which will guarantee protection from data loss. I will easily be able to switch to using university provided computers upon hardware/software failure.

The project will be built upon work that has been done in the DCML (Digital cognitive musicology lab) based in EPFL. The files are in their Github repository, and I have been granted permission to access their in-house datasets of score annotations, as well as software packages which are used to handle the data.

B.7 Supervisor Information

Peter Harrison, head of Centre for Music and Science at Cambridge, has agreed to supervise me for this. We have agreed on a timetable for supervisions for this year. I am also working with Christoph Finkensiep, a PHD student at the DCML, and originator of the proto-voice model. Professor Larry Paulson has agreed to be the representative university teaching officer.

Bibliography

[1] Chen, T.-P., and Su, L. Functional Harmony Recognition of Symbolic Music Data with Multi-task Recurrent Neural Networks, Sept. 2018.

[2] Chen, T.-P., and Su, L. Harmony Transformer: Incorporating Chord Segmentation into Harmony Recognition. In *International Society for Music Information Retrieval Conference* (2019).

[3] Cohen, D. E. "The Imperfect Seeks Its Perfection": Harmonic Progression, Directed Motion, and Aristotelian Physics. *Music Theory Spectrum 23*, 2 (Oct. 2001), 139–169.

[4] Davidson-Pilon, C. Bayesian Methods for Hackers, Mar. 2023.

[5] Deng, J., and Kwok, Y.-K. Large vocabulary automatic chord estimation using bidirectional long short-term memory recurrent neural network with even chance training. *Journal of New Music Research 47*, 1 (Jan. 2018), 53–67.

[6] Ebcioğlu, K. An Expert System for Harmonizing Four-Part Chorales. *Computer Music Journal 12*, 3 (1988), 43–51.

[7] Feisthauer, L., Bigo, L., Giraud, M., and Levé, F. Estimating keys and modulations in musical pieces. In *Sound and Music Computing Conference (SMC 2020)* (June 2020).

[8] Finkensiep, C. *The Structure of Free Polyphony.* PhD thesis, EPFL, Lausanne, 2023.

[9] Finkensiep, C., Ericson, P., Klassmann, S., and Rohrmeier, M. Chord Types and Ornamentation - A Bayesian Model of Extended Chord Profiles. *Open Research Europe* (Apr. 2023).

[10] Finkensiep, C., and Rohrmeier, M. Modeling and Inferring Proto-voice Structure in Free Polyphony. In *Proceedings of the 22nd ISMIR Conference* (Online, Nov. 2021).

[11] Gjerdingen, R. Cognitive Foundations of Musical Pitch Carol L. Krumhansl. *Music Perception: An Interdisciplinary Journal 9* (July 1992), 476–492.

[12] Goodman, J. Semiring Parsing. *Computational Linguistics 25*, 4 (1999), 573–606.

[13] Gotham, M., Kleinertz, R., Weiss, C., Müller, M., and Klauk, S. What if the 'When' Implies the 'What'?: Human harmonic analysis datasets clarify the relative role of the separate steps in automatic tonal analysis. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR 2021, Online, November 7-12, 2021* (2021), J. H. Lee, A. Lerch, Z. Duan, J. Nam, P. Rao, P. van Kranenburg, and A. Srinivasamurthy, Eds., pp. 229–236.

[14] Granroth-Wilding, M. Harmonic Analysis of Music Using Combinatory Categorial Grammar. *The University Of Edinburgh* (2013).

[15] Harasim, D. Harmonic Syntax in Time: Rhythm Improves Grammatical Models of Harmony. In *Proceedings of the 20th ISMIR Conference* (2019), T. J. O'Donnell and M. A. Rohrmeier, Eds., ISMIR.

[16] Humphrey, E. J., and Bello, J. P. Four Timely Insights on Automatic Chord Estimation.

[17] Johannes, H. Ms3 - Parsing MuseScore 3. https://github.com/johentsch/ms3, 2021.

[18] Kidney, D. O., and Wu, N. Algebras for weighted search. *Proceedings of the ACM on Programming Languages 5*, ICFP (Aug. 2021), 1–30.

[19] Koops, H. V., de Haas, W. B., Bransen, J., and Volk, A. Automatic chord label personalization through deep learning of shared harmonic interval profiles. *Neural Computing and Applications 32*, 4 (Feb. 2020), 929–939.

[20] Koops, H. V., de Haas, W. B., Burgoyne, J. A., Bransen, J., Kent-Muller, A., and Volk, A. Annotator subjectivity in harmony annotations of popular music. *Journal of New Music Research 48*, 3 (May 2019), 232–252.

[21] Korzeniowski, F., and Widmer, G. Improved Chord Recognition by Combining Duration and Harmonic Language Models, Aug. 2018.

[22] Krumhansl, C. L., and Kessler, E. J. Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys. *Psychological Review 89* (1982), 334–368.

[23] Lerdahl, F., and Jackendoff, R. *A Generative Theory of Tonal Music*, repr. ed. MIT Press, Cambridge, Mass., 2010.

[24] Marsden, A. Schenkerian Analysis by Computer: A Proof of Concept. *Journal of New Music Research 39*, 3 (Sept. 2010), 269–289.

[25] Masada, K., and Bunescu, R. Chord Recognition in Symbolic Music: A Segmental CRF Model, Segment-Level Features, and Comparative Evaluations on Classical and Popular Music, Oct. 2018.

[26] Mauch, M., Mullensiefen, D., Dixon, S., and Wiggins, G. Can Statistical Language Models be used for the Analysis of Harmonic Progressions?

[27] Maxwell, H. J. An expert system for harmonizing analysis of tonal music. In *Understanding Music with AI: Perspectives on Music Cognition*. MIT Press, Cambridge, MA, USA, Aug. 1992, pp. 334–353.

[28] McLeod, A., and Rohrmeier, M. A Modular System for the Harmonic Analysis of Musical Scores using a Large Vocabulary. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR 2021, Online, November 7-12, 2021* (2021), J. H. Lee, A. Lerch, Z. Duan, J. Nam, P. Rao, P. van Kranenburg, and A. Srinivasamurthy, Eds., pp. 435–442.

[29] Mearns, L. The Computational Analysis of Harmony in Western Art Music.

[30] Neuwirth, M., Harasim, D., Moss, F. C., and Rohrmeier, M. The Annotated Beethoven Corpus (ABC): A Dataset of Harmonic Analyses of All Beethoven String Quartets. *Frontiers in Digital Humanities 5* (July 2018), 16.

[31] Ni, Y., Mcvicar, M., Santos-Rodriguez, R., and De Bie, T. An end-to-end machine learning system for harmonic analysis of music, July 2011.

[32] Pardo, B., and Birmingham, W. P. Algorithms for Chordal Analysis. *Computer Music Journal 26*, 2 (June 2002), 27–49.

[33] Pearl, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. The Addison-Wesley Series in Artificial Intelligence. Addison-Wesley Pub. Co, Reading, Mass, 1984.

[34] Pickens, J. *Harmony Modeling for Polyphonic Music Retrieval*. PhD thesis, University of Massachusetts, 2004.

[35] Quinn, I. Are Pitch-Class Profiles Really "Key for Key"? *Zeitschrift der Gesellschaft für Musiktheorie [Journal of the German-Speaking Society of Music Theory] 7* (Jan. 2010).

[36] Radicioni, D. P., and Esposito, R. BREVE: An HMPerceptron-Based Chord Recognition System. In *Advances in Music Information Retrieval*, J. Kacprzyk, Z. W. Raś, and A. A. Wieczorkowska, Eds., vol. 274. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 143–164.

[37] Raffel, C., McFee, B., Humphrey, E. J., Salamon, J., Nieto, O., Liang, D., and Ellis, D. P. W. Mir_eval: A Transparent Implementation of Common MIR Metrics. In *Proceedings of the 15th International Conference on Music Information Retrieval* (2014).

[38] Raphael, C., and Stoddard, J. Functional Harmonic Analysis Using Probabilistic Models. *Computer Music Journal 28*, 3 (Sept. 2004), 45–52.

[39] Rocher, T., Robine, M., Hanna, P., and Strandh, R. Dynamic Chord Analysis for Symbolic Music.

[40] Sapp, C. Visual hierarchical key analysis. *Computers in Entertainment 3* (Oct. 2005), 1–19.

[41] Sapp, C. 6 Computational Chord-Root Identification in Symbolic Musical Data: Rationale, Methods, and Applications.

[42] Sidorov, K., Jones, A., and Marshall, D. MUSIC ANALYSIS AS A SMALLEST GRAMMAR PROBLEM.

[43] Temperley, D. An Algorithm for Harmonic Analysis. *Music Perception 15*, 1 (Oct. 1997), 31–68.

[44] Temperley, D. A Bayesian Approach to Key-Finding. In *Music and Artificial Intelligence*, G. Goos, J. Hartmanis, J. van Leeuwen, C. Anagnostopoulou, M. Ferrand, and A. Smaill, Eds., vol. 2445. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 195–206.

[45] Temperley, D. A Bayesian Approach to Key-Finding. In *Music and Artificial Intelligence* (Berlin, Heidelberg, 2002), C. Anagnostopoulou, M. Ferrand, and A. Smaill, Eds., Lecture Notes in Computer Science, Springer, pp. 195–206.

[46] Temperley, D. A Unified Probabilistic Model for Polyphonic Music Analysis. *Journal of New Music Research 38*, 1 (Mar. 2009), 3–18.

[47] Volk, A. Improving Audio Chord Estimation by Alignment and Integration of Crowd-Sourced Symbolic Music. 141–155.

[48] Winograd, T. Linguistics and the Computer Analysis of Tonal Harmony. *Journal of Music Theory 12*, 1 (1968), 2–49.