Doctoral Dissertation

Exploring Effective Feature Combinations to Define Harmonic Distance Models

Hiroyuki Yamamoto

Supervisor: Ryuhei Uehara

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

September, 2025

Abstract

Music is a universal culture of humankind that is believed to have existed since the prehistoric ages, and it has become an indispensable part of life for many people even today. The fact that many people spend a great deal of time, effort, and money on music suggests its considerable importance. Music can be discussed from various aspects, but three elements are generally considered important for its composition, especially in Western music: rhythm, melody, and harmony. This study focuses on harmony, particularly in tonal music.

There have been many attempts to analyze the harmonic structure in tonal music. One of the standard methods is to use a distance model to represent the structure of harmony. Broadly speaking, there are theory-based models and statistical/machine learning-based models, each of which has its own advantages and disadvantages. In general, theory-based models are more interpretable and easier to use for human analysis, understanding, and education of music. On the other hand, however, they also have drawbacks: they are not well supported by actual data, they are not easy to fine-tune due to differences in types or genres of music, and they are also not easy to modify or expand the structure without specialized knowledge. We therefore aim to integrate theory-based and data-based methods to address these weaknesses while preserving interpretability.

First, we propose a framework in which we construct, train, and evaluate a variety of distance functions that share the same domain and range as TPS. We define three basic harmonic features (mode, tonic, and degree), and utilize these to define various distance functions by combining them, and then we evaluate all simple combinations exhaustively. We then train and evaluate each function through the task of key estimation. This entire process is to provide a perspective from which to examine how well various features and their combinations can represent harmony using our distance functions, which are themselves distance models, as probes. We show that very simple functions, i.e., functions with a very small number of effective parameters (#EP), can achieve around 80% accuracy as long as they include all important harmonic features. This accuracy score is greater than that of TPS. On the other hand, we also confirm that some aspects of TPS are indeed well designed. In addition, a learned model is compared to other theoretical and experiment-based models to verify agreement and minor differences.

This framework, however, uses the task of key estimation for a sequence of chord symbols, where the distances between a chord symbol and its candidate interpretations are not taken into account. If we consider TPS as a model that defines the distance between chord interpretations, the distance between a chord symbol and its candidate interpretation is inherently out of scope. However, since

basic space, one of the components of TPS, calculates distances by taking into account the importance of each pitch class (PC), this structure can be applied to the calculation of distance between chord symbols and chord interpretations. So we extend the framework one more step to consider the distance between chord symbols and candidate chord interpretations by comparing them on a PC-by-PC basis. The functions are then trained and evaluated in the task of key:degree estimation from a sequence of chroma vectors. This allows the importance (or distance) of each PC to be obtained from the data. Experiments showed that the structure of the basic space and the assigned values were quite appropriate, but the performance can be further improved by adding the distinction of major/ minor scales.

Keywords: Harmony Analysis; Distance Model; Probabilistic Model; Tonal Music; Machine Learning

Acknowledgments

First and foremost, I would like to express my deepest gratitude to Professor Satoshi Tojo of Asia University, whose insightful guidance, constructive feedback, and continuous support throughout my doctoral research. His mentorship has greatly influenced not only the development of this dissertation but also my grouth as a researcher.

I also thank Professor Ryuhei Uehara of Japan Advanced Institute of Science and Technology (JAIST), my official academic advisor, for his guidance and support during the course of my doctoral work. In addition to fulfilling the formal advisory role, he provided helpful advice and feedback on various aspects of my research.

I am sincerely grateful to Professor Masashi Unoki of JAIST for his valuable guidance on the minor research project. His expertise greatly contributed to shaping this line of research.

I am indebted to Professors Kunihiko Hiraishi of JAIST, Kazuhiro Ogata of JAIST, and Tetsuro Kitahara of Nihon University for serving as reviewers of my doctoral dissertation. Their constructive feedback and thoughtful questions during the review process were instrumental in refining the final version of this work.

I would also like to extend my appreciation to Dr. Yui Uehara, currently an Assistant Professor at Kanagawa University and a former member of Tojo Laboratory, for her generous support and advice on both academic and practical matters.

Finally, I wish to thank everyone who supported me during my Ph.D. study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Harmony is one of the most fundamental components of music [35], and, like natural language, there must be some kinds of syntax in harmonic sequences [2]. In tonal music, each chord does not exist independently but is arranged with respect to the context. So, if put in a different context, the same set of tones (or notes) may arouse a different interpretation for us and therefore can be given a different interpretation as a chord. For example, the C major triad can be interpreted as C:I, a:III, G:IV, F:V, e:VI, and d:VII, but if it appears in a chord sequence like G $\rightarrow$ C $\rightarrow$ D $\rightarrow$ G, it will most likely be considered G:IV. To determine an interpretation, we need to take the relationship between chords into account, based on a reliable theory of harmonic structure. One way to do this is to introduce a distance over musical elements.

There have been a lot of approaches to introduce a notion of distance to express harmonic features. Heinichen [14] and Kellner [16] attempted to represent the key relationship in circular spatial models based on the circle of fifths, and Weber [40] and Schoenberg [33] proposed related spatial models. Riemann [28] applied the Tonnetz, which had been invented by Euler [10] as a way of representing just intonation, to analyze harmonic relationships from the viewpoint of pitch class (PC). This model has inspired a number of recent studies (e.g., [6, 15, 36, 37, 25]). Longuet-Higgins [18, 19] proposed the "tonal space" as a three-dimensional space in which tones in a relationship of fifth, major third, and octave are adjacent to each other. Hall [13] and Balzano [1] also proposed a similar model. Drobisch [9] and Shepard [31, 32] proposed methods to express a pitch as a coordinate on a three-dimensional helical structure to represent two types of proximity: the proximity in terms of PC (or octave) and the proximity in terms of fundamental log-frequency. Chew [5] then took advantage of this helix structure to obtain the coordinates of

triads and keys by theoretical calculation. The model of Bharucha & Krumhansl [2] investigated the perceived similarity in diatonic triads within a key. Similarly, Krumhansl [17] used PC proximity, chordal proximity, and regional proximity from empirical data in their respective spatial models. Lerdahl [20] introduced another geometrical model, Tonal Pitch Space (TPS), which also takes into account the same three proximities as [17], but this model, TPS, uses the circle of fifths and its variant to define the proximities between regions and between chords, and also incorporates a hierarchical framework proposed by Deutsch & Feroe [8] to consider the importance of each PC. More recently, there has been a lot of research on neural network methods that utilize large-scale data. Madjiheurem et al. [22] applied *word2vec* [24], a method in the field of natural language processing, to musical chords to obtain a vector representation of the chords. Harmonic analysis using the Transformer has also been attempted [4], and the attention map that appears here may also be seen as a kind of proximity expression.

Some of these distance models are acquired from experimental data, while some are derived rather theoretically. Models belonging to the latter lineage have the advantage of superior interpretability, making them easier to apply to human analysis, understanding, and teaching of music. On the other hand, they also have disadvantages: they have limited empirical support; they are generally inflexible, making it difficult to represent diversity in genres, etc.; they are not easy to redesign or extend their structures and rules, requiring specialized knowledge. TPS is one of the latter models. Although it seems theoretically convincing, the procedure to compute the distance and assigned values are not defined in an objective manner. Randall & Khan [27] have compared TPS and the model of Bharucha & Krumhansl [2], which belongs to a data-driven formalism, and tried to create a new distance model by combining both models.

In this study, we also explore the way to integrate empirical data into the theory of TPS based on Yamamoto & Tojo [43, 44]. We propose a framework in which we construct a variety of distance models (expressed as functions), which have the same domain and range as TPS (i.e., both receive a pair of key:degree information as *chord interpretations*, and return a numeric value), as combinations of harmonic features (mode, tonic, and degree), then train them by annotated data, and compute the most plausible interpretation paths. This will preserve the interpretability that characterizes theory-based models, while addressing the weakness of data support and the difficulty of fine-tuning to individual genres or types (although the data used in this study's experiments are mainly for the so-called period of common practice [26], the data could be swapped to accommodate other genres or types) by utilizing machine learning, and the difficulty of modifying and expanding the model by defining various ways to combine features and functions. TPS claims that the shortest path yields the most natural interpretation,

so, conversely, a model that gives the shortest path for the best interpretation with higher accuracy could be considered to better capture the harmonic structure that Lerdahl was aiming to determine with TPS. Therefore, as the final step in our proposed framework, we compare the effectiveness of harmonic feature combinations through the accuracy of key estimation. And furthermore, we extend this proposed framework incorporating a structure that computes the distance between a chord symbol and its interpretation.

1.2 Research Objective

In this study we generalize the idea of TPS and propose a new distance model in which we can construct various distance functions by combining harmonic features. Then we explore what features are important, what combinations are effective, and what distance values should be assigned for defining distance functions of harmony. TPS was given its structure and values theoretically, but the proposed framework enables our functions to learn them from the data. Effectiveness is also verified using data (test sets). The proposed distance model is compatible with TPS, so it can be compared with TPS, and also can be combined with (part of) TPS to build another distance function.

We show that very simple functions, i.e., functions with a very small number of effective parameters (#EP), can achieve around 80% accuracy as long as they include all important harmonic features. This accuracy score is greater than that of TPS. On the other hand, we also confirm that some aspects of TPS are indeed well designed.

This framework, however, uses the task of key estimation for a sequence of chord symbols for learning and evaluation, where the distances between a chord symbol and its candidate interpretations are not taken into account. Since TPS is a model that defines the distance between chord interpretations, the distance between a chord symbol and its candidate interpretation is inherently out of scope. So we extend the framework one more step. We apply basic space, one of the components of TPS, to define a method of calculating the distance between chord symbols and candidate interpretations by comparing them on a PC-by-PC basis. This is then learned and evaluated by a key and chord interpretation estimation task. This allows the importance (distance) of each PC, like basic space, to be obtained from the data.

1.3 Dissertation Outline

The rest of this dissertation is organized as follows. We first review the basics and some applications of TPS in Chapter 2. We then give a detailed explanation of the structure of our distance models (Chapter 3) and the learning strategy (Chapter 4). Thereafter, we present and discuss the results of our experiments in Chapter 5. In the next chapter we explain a way to extend the framework to incorporate the distance between chords and their interpretations and show the results of experiments on the extended framework (Chapter 6). And finally conclude in Section 7.

Note that this dissertation is primarily based on my previously published articles [45, 46].

Chapter 2

Background

2.1 Tonal Pitch Space

2.1.1 Overview

Tonal Pitch Space (TPS) is a music model for the quantitative harmony analysis proposed by Fred Lerdahl [20]. It is proposed to complement Lerdahl and Jackendoff's "Generative Theory of Tonal Music (GTTM)" [21], which applies a generative grammar to extend Schenkerian theory. The proposed distance model is compatible with the TPS, so it can be compared with the TPS, and also can be combined with (part of) TPS to build another distance model.

In this model, a chord can be interpreted in multiple key:degree pairs (e.g., interpretation of C major triad are as follows: C:I, a:III, G:IV, F:V, e:VI, and d:VII) and TPS defines a distance between every pair of these chord interpretations. TPS also claims the principle of the shortest path, i.e., when the distance of two chord interpretations in TPS is small, the transition between them should sound natural to us. By means of this, we can determine the most plausible interpretation pair as the pair which achieves the shortest distance.

A distance in TPS is defined as a combination of three elements explained below.

2.1.2 Regional Distance

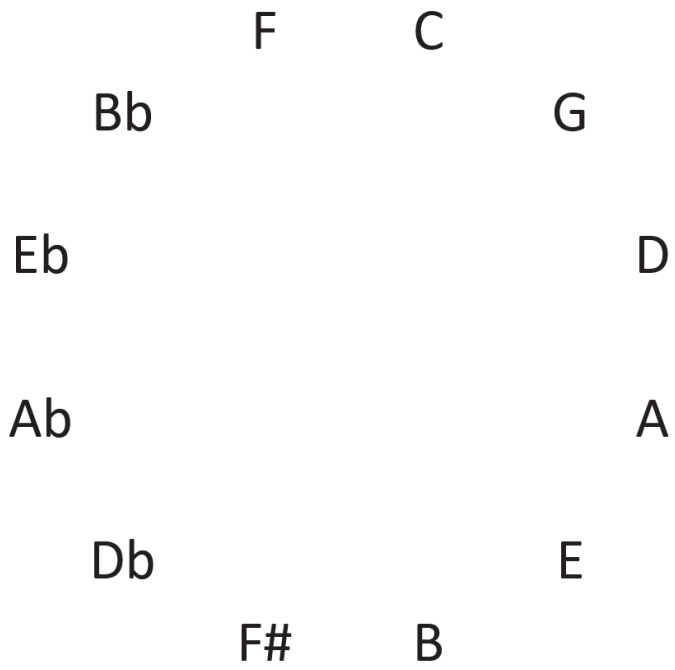

Figure 2.1: The regional circle-of-fifths

The first element is the distance between keys, and is defined as the length of the shortest arc on the regional circle of fifths (Figure 2.1). This distance cannot be calculated between major and minor keys, so one must convert either key to the key with the same key signature. For example, to calculate the regional distance between G major and A minor, we must convert G major to E minor, or A minor to C major, then we can calculate the distance, 1 in this case, from the regional circle-of-fifths. We use a function **region** to express this distance, so, the regional distance between chord interpretations x and y is expressed as $\mathbf{region}(x, y)$.

2.1.3 Chordal Distance

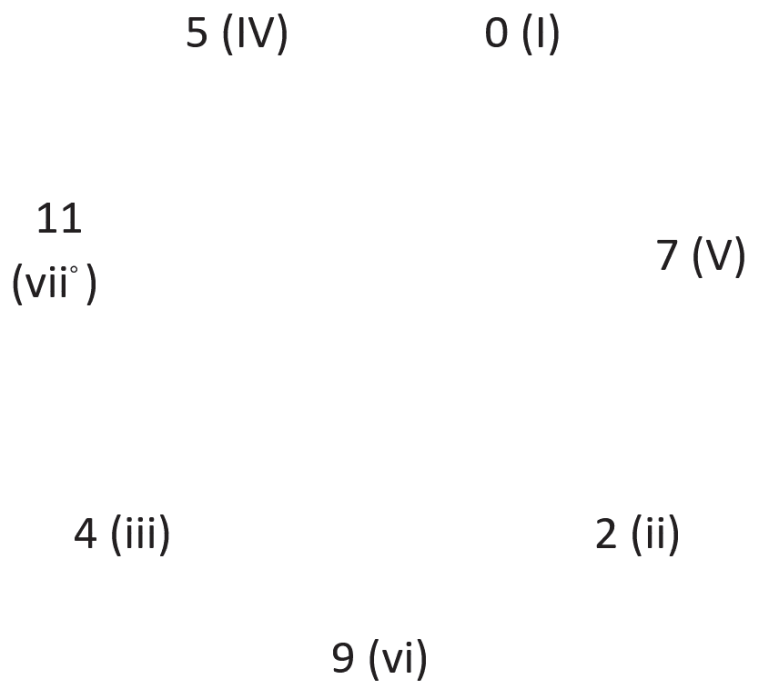

Figure 2.2: The chordal circle-of-fifths

Along with the regional circle-of-fifths, TPS defines chordal circle-of-fifths (Figure 2.2), in which degrees are arranged cyclically in the order of $(\mathrm{I}, \mathrm{V}, \mathrm{ii}, \mathrm{vi}, \mathrm{iii}, \mathrm{vii}^\circ, \mathrm{iv})$. The second element is the distance between root notes, and is defined as the length of the shortest arc on the chordal circle-of-fifths. Note that though this is also called "circle-of-fifths", not all adjacent elements are of perfect fifths. We use a function **chord** to express this distance.

2.1.4 Distance based on the Basic Space

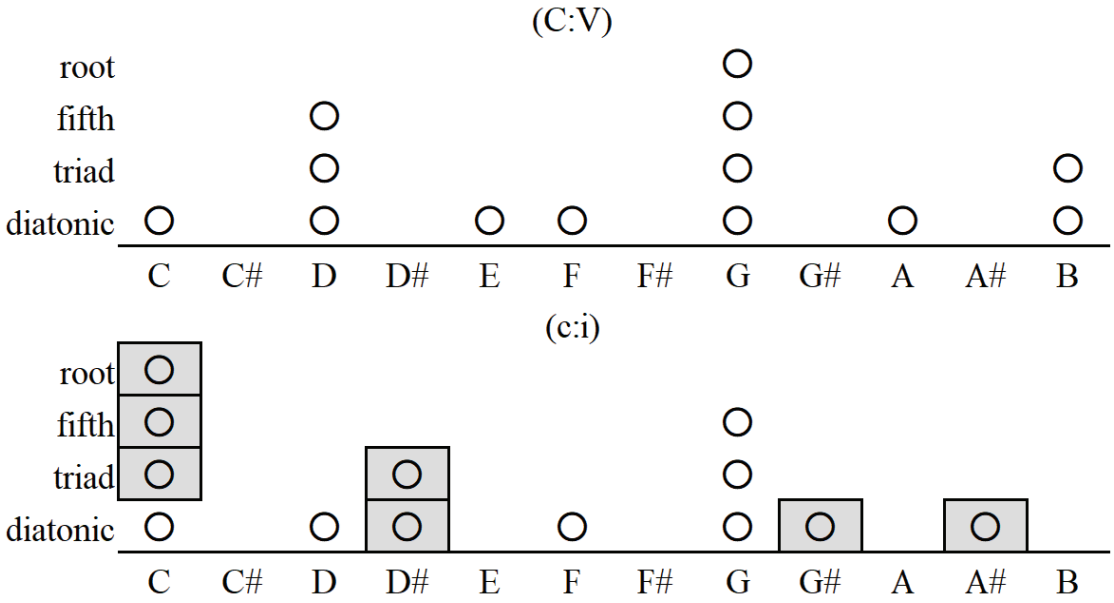

Figure 2.3: An example of basic-space, from C:V to c:i.

The last element is based on a structure called basic space, which concerns the importance of each pitch class (PC) relating to the chord interpretations. Basic space is composed of five levels (i.e., *root, fifth, triad, diatonic*, and *chromatic*) and each level contains PCs reflecting the chord interpretation. The levels are arranged in order of the importance of a PC to the chord, level *root* being the most important. Figure 2.3 shows an example from C:V to c:i. For each chord interpretation, the root PC of the chord is *root* level (which is indicated by four vertical ∘'s), the fifth PC is *fifth* level (three ∘'s), the third PC is *triad* level (two ∘'s), and every other diatonic PCs is *diatonic* level (one ∘).[1] Then the basic space-space distance between two chord interpretations is defined as the number of ∘'s that exist only in the destination. In Figure 2.3, there are seven missing circles (which are highlighted), therefore, the distance in this case is 7. We use a function **basicspace** to express this distance.

[1]We omit the *chromatic* level in Figure 2.3 because it does not affect the results.

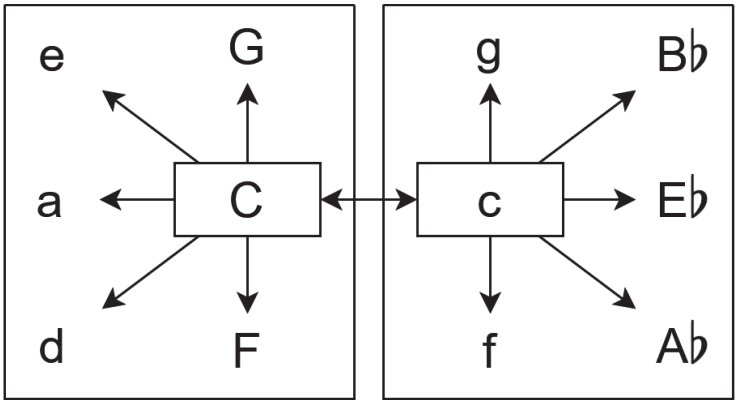

Figure 2.4: Related keys from C and c, i.e., rel(C) and rel(c).

2.1.5 Distance within Related Keys

The distance between chord interpretation x and y, denoted by $\mathbf{tps}(x, y)$, is defined as the unweighted linear sum of these three functions.

$$\mathbf{tps}(x, y) = \mathbf{region}(x, y) + \mathbf{chord}(x, y) + \mathbf{basicspace}(x, y). \tag{2.1}$$

However, Equation (2.1) is applicable only when x and y are in *related keys*, which are defined as Equation (2.2) also illustrated in Figure 2.4.

$$rel(R) = \begin{cases} \{\text{I}, \text{i}, \text{ii}, \text{iii}, \text{IV}, \text{V}, \text{vi}\} & \text{if } R \text{ is a major key} \\ \{\text{i}, \text{I}, \flat\text{III}, \text{iv}, \text{v}, \flat\text{VI}, \flat\text{VII}\} & \text{otherwise} \end{cases} \tag{2.2}$$

where Roman numerals in this equation and figure mean the keys with the tonic being the degree in key R, and upper or lower cases mean major or minor keys. The function rel maps a key to a set of related keys. For example, rel(F) is the set {F, f, g, a, B♭, C, d}.

2.1.6 Distance across Related Keys

If we interpret a transition between keys which are not in related keys (i.e., *distant keys*) as a direct modulation, it can sound unnatural to us. Instead, in TPS, it is interpreted as a combination of modulations between related keys. For example, modulation from C major to D major should be interpreted as modulations C major → G major → D major, instead of just C major → D major. Note that , there are infinite combinations to realize this, so we must choose the one that achieves the shortest total distance. Also note that, in all passing keys (e.g., G major, in this example) the degrees are set to be one. Therefore, if x and y are

distant keys, distance between x and y must be computed as

$$\begin{aligned}\mathbf{tps}(x, y) &= \min_{S \in rel(R_x), T \in rel(R_y)} (\mathbf{tps}(x, S:\mathrm{I}) + \Delta(S, T) + \mathbf{tps}(T:\mathrm{I}, y)) \\ \Delta(S, T) &= \min_{U \in rel(S)} (\mathbf{tps}(S:\mathrm{I}, U:\mathrm{I}) + \Delta(U, T))\end{aligned} \tag{2.3}$$

where R_z stands for the key out of the key:degree, denoted by z, and R:I is key R's tonic.[2] In other words, the transition from x to y is considered as a combination of transitions within related keys, and the overall distance is the shortest total distance of the transitions.

Although the forms of the formula look quite different depending on whether they are within related keys or not, given Equation (2.3) is composed of the sum of several Equation (2.1)s, both of them can be thought of as the sum of three subfunctions (i.e., **region**, **chord**, and **basicspace**). Therefore, we can rewrite the distance as follows.

$$\mathbf{tps}(x, y) = \mathbf{tps\_region}(x, y) + \mathbf{tps\_chord}(x, y) + \mathbf{tps\_basicspace}(x, y) \tag{2.4}$$

where **tps_region**, **tps_chord**, and **tps_basicspace** are the sum of all occurrences of **region**, **chord**, and **basicspace**, respectively.

2.2 Former Approaches based on TPS

2.2.1 Approach by Sakamoto et al. (2016)

Sakamoto, Arn, Matsubara, & Tojo [30] have proposed a method that utilizes the distance defined by TPS and the principle of the shortest path to find the most plausible interpretation of a given chord sequence.

[2]Note that tonic in this case is not necessarily a major chord.

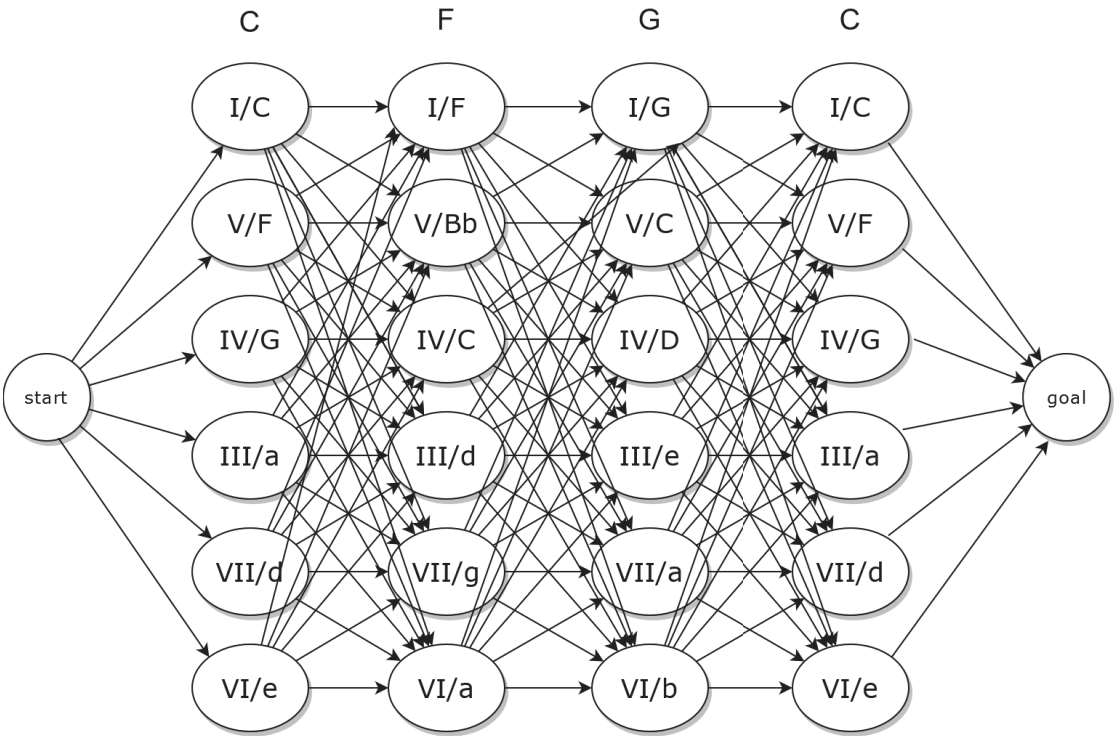

Figure 2.5: An interpretation graph with six layers.

Their method works as follows.

1. Receive a sequence of chord symbols as input.

2. Enumerate possible chord interpretations for each chord symbol, and generate nodes based on them. Also, place the nodes "start" and "goal" as dummy nodes at the beginning and the end.

3. Draw directed edges so that the node sets corresponding to adjacent chord symbols are fully connected and along the order of the chord symbol sequence from the first to the last. The graph generated given a sequence C $\rightarrow$ F $\rightarrow$ G $\rightarrow$ C is shown in Figure 2.5. We call these graphs *interpretation graphs*.

4. Assign a TPS distance between its starting and ending chord interpretation for each edge. Note that the edges whose one end is "start" or "goal" are set to 0.

5. The shortest path from "start" node to "goal" node is searched for by the Dijkstra's algorithm. The node sequence on this path is considered the op-

timal interpretation for the input sequence. For the graph shown in Figure 2.5, one of the optimal interpretations is $\mathrm{C} : \mathrm{I} \to \mathrm{IV} \to \mathrm{V} \to \mathrm{I}$.[3]

2.2.2 Approach by Yamamoto et al. (2020)

Yamamoto, Uehara, & Tojo [42] proposed the following three extensions on Sakamoto et al.'s method (including TPS) with jazz in particular in mind.

extension (i) **Tetrads and Scales**
They extended the method to include some commonly used chord types in jazz, like tetrads, and a distinction of three minor scales.

extension (ii) **ϵ-transitions**
They also introduced an ϵ-transition to express a pivot chord, which is a chord that bridges a key modulation and needs to be interpreted in both keys. To bring this into calculation, they modified the interpretation graph duplicating every chord interpretation candidate so that each chord can have at most two interpretation at once without losing applicability of Viterbi algorithm [39]. In Figure 2.6, all diagonal arrows within rectangles are ϵ-transitions. They gave it a fixed cost of 0.5.

extension (iii) **Cadential Shortcuts**
Thirdly, to handle chord sequence of more than two chords, they extended the interpretation graph further to add new edges, which they call cadential shortcuts. Cadential shortcuts have half the value (i.e., distance) of the original routes so that the paths which include them will more likely be chosen as the shortest paths. This modification also retains the applicability of the Viterbi algorithm. In Figure 2.6, a cadential shortcut for $\mathrm{C} : \mathrm{II}^7_{\mathrm{m}} \to \mathrm{V}^7 \to \mathrm{I}^7_{\mathrm{M}}$ is shown as a dashed arrow.

With the above extensions, the interpretation graph becomes as shown in Figure 2.6.

[3] An upper-case Roman numeral following a key and colon (:) represents the scaled degree of the root of the chord. Note that all such Roman numerals do not reflect the distinction of chord qualities.

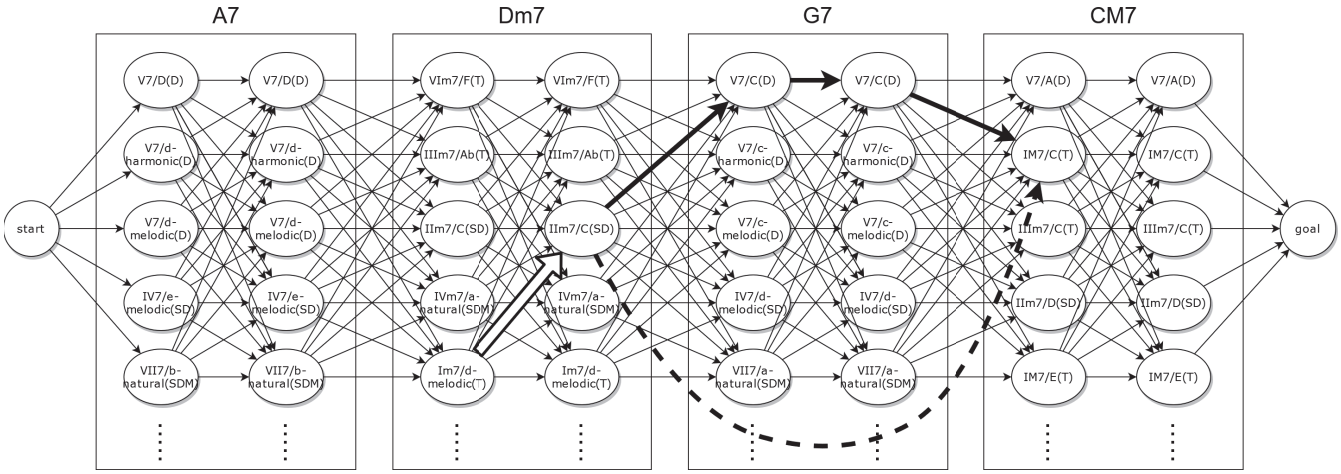

Figure 2.6: Extended interpretation graph

They implemented the method in a program and conducted experiments. One of the problem with Sakamoto et al.'s method was that their method of calculating distances was too simple and generated a large number of shortest paths (i.e., the result was highly ambiguous), but there was considerable improvement in this respect. Moreover, as shown in Figure 2.7 and 2.8, the expressive power of interpretation has increased, allowing for more detailed analyses.

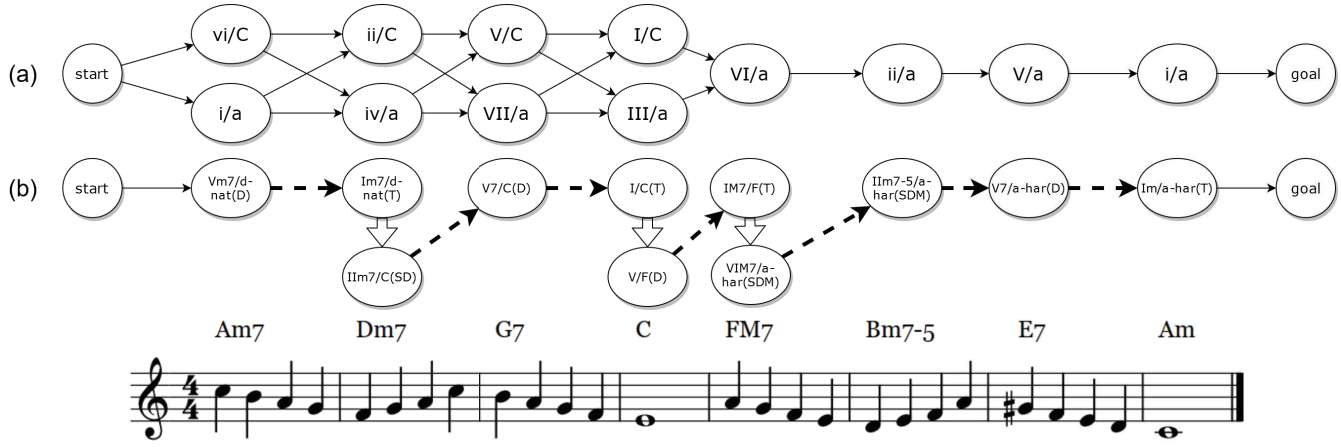

Figure 2.7: Analyses of "Fly Me to the Moon" (a) Sakamoto et al.'s method (b) proposed method

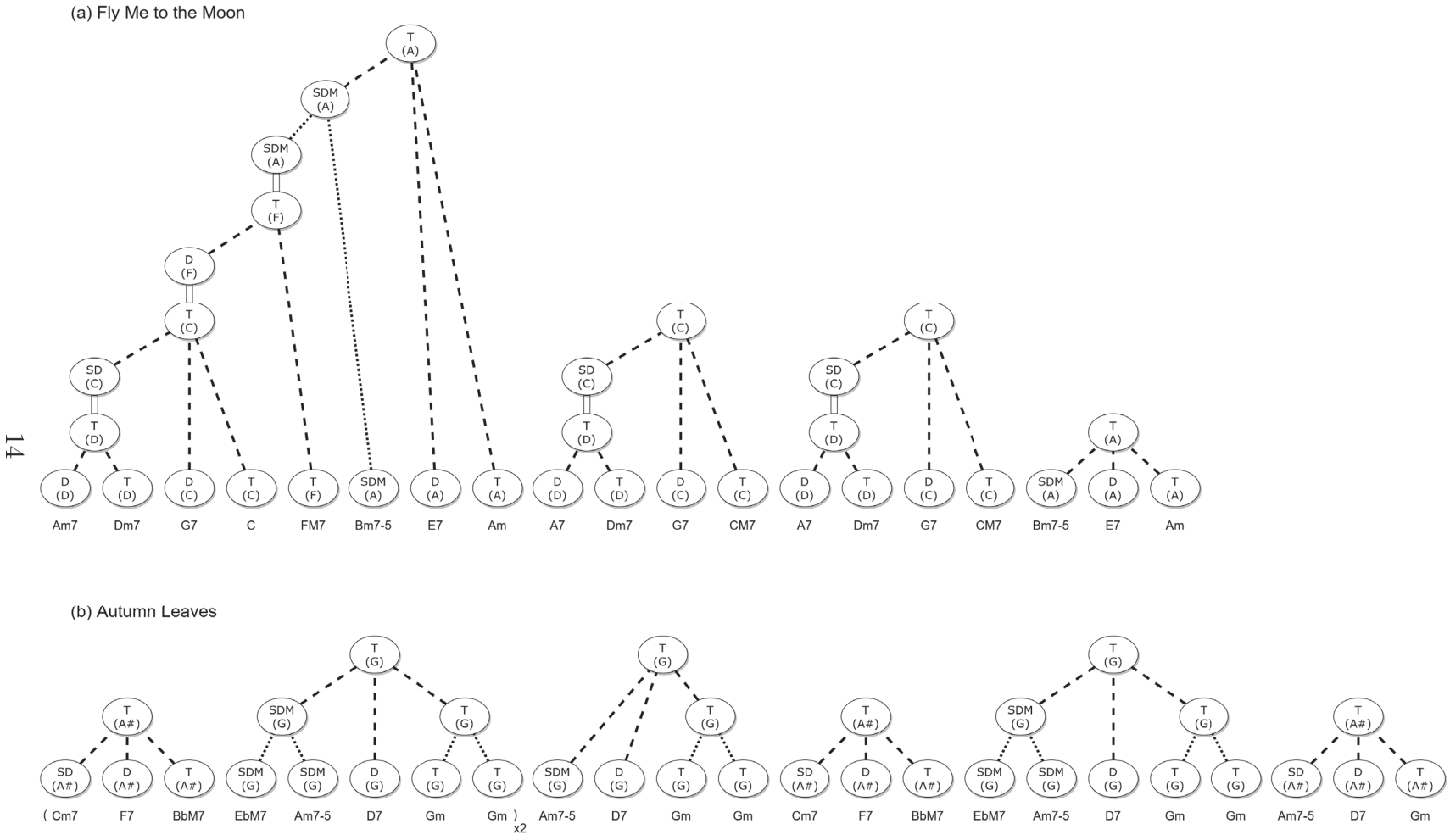

Figure 2.8: Tree representations of (a) “Fly Me to the Moon” (b) “Autumn Leaves”

2.2.3 Other approaches

In addition to the above studies, various other TPS-based methods have been proposed.

Based on the key profiles of Temperley [34] alongside TPS, Catteau, Martens, & Leman [3] have proposed a framework to estimate keys and chords from audio data by defining probabilities concerning chords, scales, and chroma vectors and to find the most probable path of chords and scales with the use of Viterbi search. Rocher, Robine, Hanna, & Oudre [29] also used Temperley's key profiles and TPS to estimate individual chords and keys from audio data. Matsubara, Kodama, & Tojo [23] have proposed to restrict the minor scale to the harmonic one to improve cadence detection by avoiding ambiguity in chord interpretations. Also, Yamaguchi & Sugamura [41] have proposed to extend the basic space with an additional layer particularly designed to express seventh chord.

Yamamoto & Tojo [43] have tried to generalize TPS with a learnable structure, based on the method of Sakamoto et al. [30]. They proposed several functions called 'distance elements (DEs)' and a way to train them with annotated data. Their best function achieved over 86% accuracy on a key estimation task using the dataset published by Gotham, Kleinertz, Weiss, Müller, & Klauk [11] while the original TPS scored about 40%, and they also found a model with just 58 learnable parameters could achieve more than 80%. However, they defined only 15 DEs (including those of the original TPS') and some combinations of them. Yamamoto & Tojo [44] then introduced a way to restructure DEs (and their combinations) in a more systematic way so that they can compare all simple combinations of the basic features, and show there are more efficient uses of features to define distance functions. In this study, we improve their way of restructuring distance models (i.e., DEs) so that it allows us to explore more comprehensively and give detailed analyses.

Chapter 3

Proposed Distance Model

In this section, we propose a way to construct functions that return numerical distance for a given chord interpretation pair. At a high level, our model consists of the following steps.

1. We define various *distance functions*, combining multiple *elemental distance functions*. Let an instance of such a function be $\boldsymbol{\delta}(x, y)$, where x is a source chord interpretation and y is its following one (Figure 3.1 shows an example where $x =$ C:I and $y =$ d:III).

2. Each $\boldsymbol{\delta}$ performs, essentially, as a case distinction, and the coefficients of these cases are stored in its *parameter array*, shown by $v[i]$ where $i \in \{j \in \mathbb{Z} \mid 0 \leq j \leq n\}$ and n is determined by the way $\boldsymbol{\delta}$ is defined. The index i of $v[i]$ is chosen, dependent on two input chord interpretations x, y of $\boldsymbol{\delta}(x, y)$, and the distance results in $\boldsymbol{\delta}(x, y) = v[i]$.

3. We define a probability P of a sequence of chord interpretations (interpretation path), by converting a distance to probability by Equation (4.1).

4. The values of $v[i]$ for each $\boldsymbol{\delta}$ are obtained by maximum-likelihood estimation.

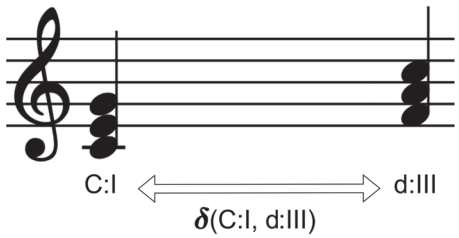

Figure 3.1: An example of two chord interpretations and the distance.

In Section 3.1, we define three basic features of chord interpretations with which the elemental distance functions, mentioned at (1) and (2) above, are defined in the following section. Elemental distance functions are the simplest form of distance functions and can be combined with other distance functions to create another distance function, as explained in Section 3.3. Each distance function assembled this way will be modeling harmonic transition in a distinct way, and can be trained, as mentioned in (3) and (4) above, as long as they have learnable parameters (this topic is covered in Section 4. Finally, Sections 3.4 and 3.5 give additional details about the functions.

3.1 Basic Features

We want our distance function to be defined as a function that gives the distance in the same space as TPS. Therefore, since TPS deals with the distance between chord interpretations, we define the basic features as those that constitute chord interpretations. Let $\mathcal{X}$ be a set of chord interpretations, each of which is represented by an ordered pair (Cartesian product) of mode[1] $\mathcal{M}$, tonic (i.e., principal PC of the key) $\mathcal{T}$, and degree $\mathcal{D}$; where

$$\begin{aligned} \mathcal{M} &= \{\text{minor}, \text{major}\} \\ \mathcal{T} &= \{\text{C}, \text{C}\sharp, \text{D}, \text{D}\sharp, \text{E}, \text{F}, \text{F}\sharp, \text{G}, \text{G}\sharp, \text{A}, \text{A}\sharp, \text{B}\} \\ \mathcal{D} &= \{\text{I}, \text{II}, \text{III}, \text{IV}, \text{V}, \text{VI}, \text{VII}\}. \end{aligned} \tag{3.1}$$

Therefore,

$$\mathcal{X} = \mathcal{M} \times \mathcal{T} \times \mathcal{D} = \{(\text{minor}, \text{C}, \text{I}), (\text{minor}, \text{C}, \text{II}), \cdots, (\text{major}, \text{B}, \text{VII})\}. \tag{3.2}$$

From now on, we will express the above three features using superscripts such as x^{mode}, x^{tonic}, and x^{degree}. In all cases, we will consider features as integer values starting from 0. For example, when $x = \text{f:III}$, the values of x^{mode}, x^{tonic}, and x^{degree} will be 0, 5, and 2, respectively.

3.2 Elemental Distance Functions

Now we define functions that receive two chord interpretations (namely, the source and destination of a transition) and return a real value for the transition distance (i.e., $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$). We introduce *elemental distance functions*, hereafter referred to as *elemental functions*, as the simplest form of distance functions. Moreover, we propose two kinds of them, learnable functions and non-learnable functions.

[1] In this research we deal only with major (Ionian) and minor (Aeolian) modes.

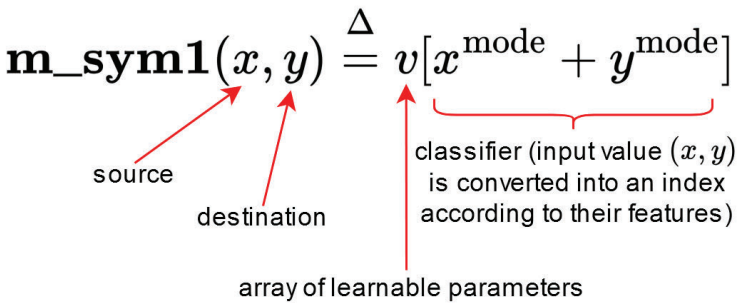

Figure 3.2: The structure of a learnable function (in this case, $\mathbf{m\_sym1}(x, y)$).

Each learnable function possesses an array of values $v[i]$ $(i = 0, \ldots, n)$, in which the results of learned values are stored. The function retrieves a value from one of the array $v[i]$, choosing such i according to the input pair of chord interpretations (Figure 3.2). For example, if we have a learned parameter set $v = [0.0, 1.5, 1.7]$[2] for **m_sym1** (Equation (3.5)), we can compute the distance $\mathbf{m\_sym1}(\mathrm{C:I, d:III})$ as follows.

$$
\begin{aligned}
&\mathbf{m\_sym1}(\mathrm{C:I, d:III}) \\
&= v[(\mathrm{C:I})^{\text{mode}} + (\mathrm{d:III})^{\text{mode}}] && \text{// from Equation (3.5)} \\
&= v[1 + 0] \\
&= 1.5 && \text{// from the learned parameter array } v
\end{aligned}
$$

A non-learnable function, on the other hand, returns predefined values and does not have learnable parameters. We can take other distance models like TPS as the elemental functions (as long as they have the same domain and range) by dealing with them as non-learnable functions.

In this study, we define three groups of learnable functions corresponding to the three basic features. In each group, we define functions according to the three *types* of information about the corresponding basic feature. That is, we consider

type (i) only source or destination (not both) features
We define this type as the minimum function. Although it seems insufficient to consider only one side when defining a distance function, they can also be combined with other functions as described below, and, in fact, some of the combinations cover all the cases that TPS can distinguish.

[2] In this example, the function distinguishes between three cases, so there are three learnable parameters (i.e., $v \in \mathbb{R}^3$).

type (ii) features of both source and destination but as an interval scale similar to **region** and **chord** in Equation (2.1), where the distances are defined based on the number of steps assuming the features to be arranged at equal intervals.

type (iii) features of both source and destination but as a nominal scale (to handle the combination of values on both sides without making any assumptions other than symmetry, which is assumed in TPS.)

And for each of these types, we define the functions as follows.

for type (i) **src** to take only the first (source) element into account

dest to take only the second (destination) element into account

for type (ii) **oneway_steps** to use the stepwise distance between source and destination while considering the direction (Figure 3.3 (a))

min_steps to use the stepwise distance between source and destination ignoring the direction (Figure 3.3 (b))

for type (iii) **sym1** to distinguish all combinations of both sides other than the direction, i.e., $\boldsymbol{\delta}(x, y) = \boldsymbol{\delta}(y, x)$ for all x and y (Figure 3.4 (a))

sym2 to distinguish all combinations of both sides other than the direction and equate all the cases where source and destination are the same, i.e., $\boldsymbol{\delta}(x, y) = \boldsymbol{\delta}(y, x)$ and $\boldsymbol{\delta}(x, x) = \boldsymbol{\delta}(y, y)$ for all x and y (Figure 3.4 (b))

In this type classification, we do not need to consider such functions that distinguish all combinations of both sides including the direction for type (iii) (Figure 3.4 (c)), since they can be achieved by the multiplication (this operation will be explained in Section 3.3) of **src** and **dest**, so we do not define them and use **src** $\times$ **dest** instead.

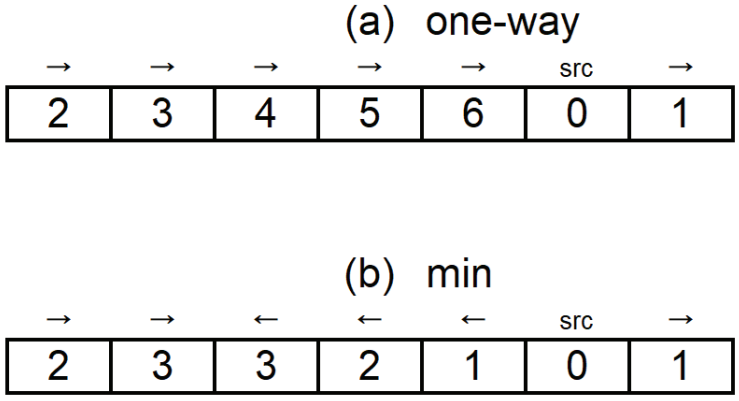

Figure 3.3: Stepwise distances when there are seven states. (a) Example of **oneway_steps**, and (b) **min_steps**.

3.2.1 Elemental Functions for Mode Feature

As the learnable functions about mode feature, we define the following four functions. For the functions of type (i), we define a function **m_src** that considers only the source side, and a function **m_dest** that considers only the destination side as follows.

$$\mathbf{m\_src}(x, y) \stackrel{\Delta}{=} v[x^{\text{mode}}], v \in \mathbb{R}^2 \tag{3.3}$$

$$\mathbf{m\_dest}(x, y) \stackrel{\Delta}{=} v[y^{\text{mode}}], v \in \mathbb{R}^2 \tag{3.4}$$

where $x, y \in \mathcal{X}$.

For type (ii), we do not define a function. This is because the mode feature has only two values, so both **oneway_steps** and **min_steps** will be the same as **sym2** in type (iii).

And for type (iii), we define **m_sym1** and **m_sym2** as follows depending on the degree of symmetry.

$$\mathbf{m\_sym1}(x, y) \stackrel{\Delta}{=} v[x^{\text{mode}} + y^{\text{mode}}], v \in \mathbb{R}^3 \tag{3.5}$$

$$\mathbf{m\_sym2}(x, y) \stackrel{\Delta}{=} v[\left|x^{\text{mode}} - y^{\text{mode}}\right|], v \in \mathbb{R}^2 \tag{3.6}$$

m_sym1 is a function that distinguishes between three cases: (major→major, minor→minor, and major↔minor), and **m_sym2** is a function that simply distinguishes whether the mode has switched or not.

3.2.2 Elemental Functions for Tonic Feature

As the learnable functions about tonic feature, we define the following four functions. For the functions of type (i) and (iii), we do not define a function because we assume the absolute position of tonic (and region) is irrelevant to chord interpretations.

For type (ii), we define **t_oneway_steps** and **t_min_steps** as follows.

$$\textbf{t\_oneway\_steps}(x, y) \triangleq v[(y^{\text{tonic}} - x^{\text{tonic}}) \mod 12], v \in \mathbb{R}^{12} \tag{3.7}$$

$$\textbf{t\_min\_steps}(x, y) \triangleq v\left[\min\begin{pmatrix}(y^{\text{tonic}} - x^{\text{tonic}}) \mod 12, \\ 12 - (y^{\text{tonic}} - x^{\text{tonic}}) \mod 12\end{pmatrix}\right], v \in \mathbb{R}^{7} \tag{3.8}$$

These are based on asymmetric and symmetric stepwise distances of tonic.

As a variant of tonic, we introduce an auxiliary feature to represent region. This can be easily constructed using tonic and mode features as follows.

$$x^{\text{region}} \triangleq \begin{cases} x^{\text{tonic}} & \text{if } x^{\text{mode}} = 1 \\ (x^{\text{tonic}} + 3) \mod 12 & \text{otherwise,} \end{cases} \tag{3.9}$$

Then we define another two functions using region instead of tonic.

$$\textbf{r\_oneway\_steps}(x, y) \triangleq v[(y^{\text{region}} - x^{\text{region}}) \mod 12], v \in \mathbb{R}^{12} \tag{3.10}$$

$$\textbf{r\_min\_steps}(x, y) \triangleq v\left[\min\begin{pmatrix}(y^{\text{region}} - x^{\text{region}}) \mod 12, \\ 12 - (y^{\text{region}} - x^{\text{region}}) \mod 12\end{pmatrix}\right], v \in \mathbb{R}^{7} \tag{3.11}$$

r_min_steps is like a learnable version of **region** in Equation (2.1). And **r_oneway_steps** is **r_min_steps** minus the symmetry assumption.

3.2.3 Elemental Functions for Degree Feature

As the learnable functions about degree feature, we define the following six functions. For the functions of type (i), we define **d_src** and **d_dest** for source and destination degrees.

$$\textbf{d\_src}(x, y) \triangleq v[x^{\text{degree}}], v \in \mathbb{R}^{7} \tag{3.12}$$

$$\textbf{d\_dest}(x, y) \triangleq v[y^{\text{degree}}], v \in \mathbb{R}^{7} \tag{3.13}$$

For type (ii), we define **d_oneway_steps** and **d_min_steps** as follows (and illustrated in Figure 3.3).

$$\textbf{d\_oneway\_steps}(x, y) \triangleq v[(y^{\text{degree}} - x^{\text{degree}}) \mod 7], v \in \mathbb{R}^{7} \tag{3.14}$$

$$\textbf{d\_min\_steps}(x, y) \triangleq v \left[\min \begin{pmatrix} (y^{\text{degree}} - x^{\text{degree}}) & \mod 7, \\ 7 - (y^{\text{degree}} - x^{\text{degree}}) & \mod 7 \end{pmatrix} \right], v \in \mathbb{R}^4 \quad (3.15)$$

These functions, especially **d_min_steps** for its symmetricity, deal with the steps in degree just like the **chord** in Equation (2.1).

Finally, for type (iii), we define **d_sym1**, and **d_sym2** as in Figure 3.4. These can be thought of as variants that allow **chord** to be learnable and also take into account the absolute position of degree.

(a)

0	1	2	3	4	5	6
1	7	8	9	10	11	12
2	8	13	14	15	16	17
3	9	14	18	19	20	21
4	10	15	19	22	23	24
5	11	16	20	23	25	26
6	12	17	21	24	26	27

d_sym1

(b)

0	1	2	3	4	5	6
1	0	7	8	9	10	11
2	7	0	12	13	14	15
3	8	12	0	16	17	18
4	9	13	16	0	19	20
5	10	14	17	19	0	21
6	11	15	18	20	21	0

d_sym2

(c)

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
42	43	44	45	46	47	48

d_asym = d_src×d_dest

Figure 3.4: Elemental functions of type (iii) when both the source and the destination have seven states (i.e., degree feature). Each table indicates the parameter indices for combinations of x^{degree} (row) and y^{degree} (column). (a) **d_sym1** ignores the direction. There are 28 independent parameters. (b) **d_sym2** ignores the direction and equates all the cases where x^{degree} and y^{degree} have the same values. There are 22 independent parameters.(c) $\textbf{d\_asym} = \textbf{d\_src} \times \textbf{d\_dest}$ distinguishes all the combinations. There are $7 \times 7 = 49$ independent parameters.

All learnable elemental functions are listed in Table 3.1.

Table 3.1: Learnable elemental functions. **type** represents the three types explained in Section 3.2.

feature	type	name	#params
mode	(i)	**m_src**	2
mode	(i)	**m_dest**	2
mode	(ii)	(= **m_sym2**)	
mode	(iii)	**m_sym1**	3
mode	(iii)	**m_sym2**	2
tonic	(i), (iii)	empty	
tonic	(ii)	**t_oneway_steps**	12
tonic	(ii)	**t_min_steps**	7
tonic	(ii)	**r_oneway_steps**	12
tonic	(ii)	**r_min_steps**	7
degree	(i)	**d_src**	7
degree	(i)	**d_dest**	7
degree	(ii)	**d_oneway_steps**	7
degree	(ii)	**d_min_steps**	4
degree	(iii)	**d_sym1**	28
degree	(iii)	**d_sym2**	22

3.2.4 Non-learnable Functions

We can use any functions as elemental functions as long as they have the form of $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$, even though they do not have learnable parameters (i.e., non-learnable). As explained above, TPS distance is composed of three sub-functions, as shown in Equation (2.4). So we can define them as three non-learnable functions, **tps_region**, **tps_chord**, and **tps_basicspace** corresponding to the terms of the equation.

3.3 Combining Elemental Functions

Elemental functions defined above can be combined to create another function (we call it a compound function, as distinguished from elemental functions). Since the elemental functions only deal with single features, this combining operation is essential to reproduce TPS.

First we define an operation that takes the product of the case divisions distinguished by the two given functions. We call this *multiplication*. This allows us to take full advantage of the functions' classification capabilities and also compose functions that encompass TPS. The number of necessary parameters is the

product of those of the two functions. For example, if we combine **m_sym1** and **t_min_steps** by *multiplication* (we denote this as '**m_sym1** $\times$ **t_min_steps**'), we need $3 \times 7 = 21$ parameters.

All learnable functions defined so far (including the compound functions) have been in the form of applying one parameter to each of the distinguished cases. This is a straightforward construction from the structure shown in Figure 3.2. However, in TPS, several distance functions calculate the distance independently and then add them together. We then define another operation that combines functions in a similar manner. We call this *addition*. The resulting function just returns the sum of the two constituent functions' results. So the number of necessary parameters is also the sum of those of the two functions. For example, if we combine **m_sym1** and **t_min_steps** by *addition* (we denote this as '**m_sym1** + **t_min_steps**'), we need $3 + 7 = 10$ parameters. Note that *addition* can be applied to non-learnable functions (this is exactly what is happening in Equation (2.4)), but *multiplication* can only be applied between learnable functions.

Because the set of functions is closed under these operations,[3] these operations can be applied recursively.[4] For example, **m_sym1** $\times$ **t_min_steps** + **d_dest** is a valid function with $3 \times 7 + 7 = 28$ parameters.

3.4 Number of Effective Parameters

We apply the functions defined above to the method proposed by Sakamoto et al. [30] to find the most plausible interpretation path as the shortest path, or *paths*, in the graph. Then the performance of a function can be evaluated by how accurately the found path(s) can predict each chord interpretation (see Section 4 for detail).

Since our purpose is to find out the most plausible interpretation path as the shortest path(s) in the interpretation graphs, any constant value can be added evenly to every distance value without changing the result. As shown in Figure 3.5, if we add a constant value to a function to define another function as **distance_b**(x, y) = **distance_a**$(x, y) + C$, the resulting shortest path(s) of these two functions are always the same. Moreover, this redundancy increases at each *addition*, because we can add any constant values to each of the constituent functions of *addition* without changing the result. In order to make it possible to compare the functions based on their efficiencies and complexities, we define the number of effective parameters (#EP) by subtracting this redundancy from the number of all parameters. For example, **m_sym1** $\times$ **t_min_steps** + **d_dest** has

[3] This is in the case of learnable functions. As mentioned above, *multiplication* is not applicable to non-learnable functions.

[4] As with normal arithmetic operations, both operations are commutative and associative, and *multiplication* has higher precedence than *addition*

28 parameters and 2 redundancy (because it has one *addition*), so its #EP is $28 - 2 = 26$.

3.5 Distance Model as a Generalized TPS

As noted above, this framework can include the functions of the original TPS as non-learnable functions. However, we can reconstruct it only with learnable functions. We can define functions with many granularities; among them, one of the finest of them is $\mathbf{m\_src} \times \mathbf{m\_dest} \times \mathbf{t\_oneway\_steps} \times \mathbf{d\_src} \times \mathbf{d\_dest}$, which distinguishes all possible cases.[5] Functions based on this form a superset of the union of other functions that are applicable to our framework, including those of the original TPS, so that we regard that our model is a generalization of TPS.

[5]To be exact, there are $2 \times 12 \times 7 \times 2 \times 12 \times 7 = 28,224$ cases in $\mathcal{X} \times \mathcal{X}$. However, as we mentioned above, we assume absolute tonic position is neutral to the interpretation, it becomes only $2 \times 12 \times 7 \times 2 \times 7 = 2,352$ cases.

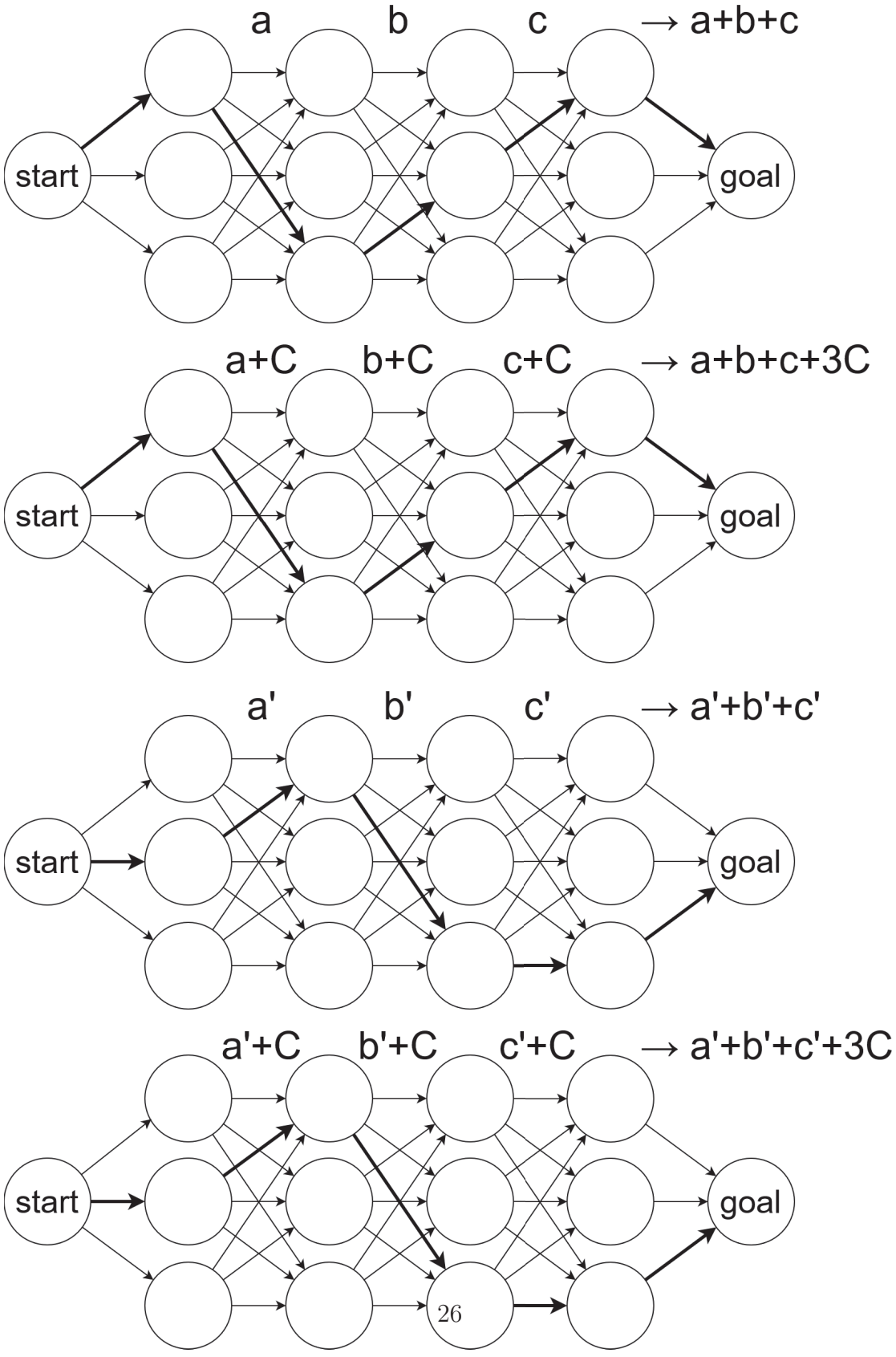

Figure 3.5: An example of adding a constant value C to every distance element. If $a + b + c$ is greater than $a' + b' + c'$, $a + b + c + 3C$ is always greater than $a' + b' + c' + 3C$.

Chapter 4

Learning Strategy

We want the functions defined in Section 3 to return such distance values that enable us to select the most plausible interpretation path in a graph as the shortest path(s). Hence we need to learn the parameters to give the ground-truth interpretation path the shortest total distance among all possible interpretation paths. In this section, we present a method for learning such parameter values.

We define notations as follows:

- $G \triangleq (V, E)$, $V \subseteq \{v | v \in \{\text{start}, \text{goal}, \mathcal{X}\}\}$, and $E = \{(u, v) | u \in G_i \text{ and } v \in G_{i+1}\}$ where G_{i+1} is the set of candidate chord interpretations for the i-th chord in the given chord sequence that has a length of $|G| - 2$. We also define $G_1 = \{\text{start}\}$ and $G_{|G|} = \{\text{goal}\}$ as a special case. We call G_i 'the i-th *layer* of G'.

- $G_{s:t} \triangleq \{(x_s, \cdots, x_t)\} \subset \mathcal{X}^{t-s+1}$: a set of all directed paths from s-th to t-th layers of G (so, $G_{s:s}$ and G_s will represent the same set). We express a node in the s-th layer as $x_s \in G_s$, the number of nodes in the s-th layer as $|G_s|$, a path from the s-th layer to the t-th layer as $x_{s:t} = (x_s, \cdots, x_t) \in G_{s:t}$, and a path from s-th layer to the $(t-1)$-th layer with x_t to be added to the tail as $x_{s:t} \in G_{s:t-1}\{x_t\}$.

- $x^*_{1:|G|} = (x^*_1, \cdots, x^*_{|G|}) \in G_{1:|G|}$: the ground-truth interpretation path.

4.1 Path Length

As explained above, [30] select the shortest path(s) in the interpretation graph (Figure 2.5) as the most plausible interpretation. They use the original TPS (Equation (2.4)) to determine each edge's weight. Then we compute the path

length of a path $x_{s:t}$ as $\mathbf{len}(x_{s:t}) \triangleq \sum_{u=s}^{t-1} \boldsymbol{\delta}(x_u, x_{u+1})$, where $\boldsymbol{\delta}$ is a meta-function which is replaced by a concrete function name defined in Section 3 but if "start" or "goal" is input for x or y, it always returns 0. We then attempt to find parameter values that minimize $\mathbf{len}(x_{1:|G|})$ for the ground-truth path, $x^*_{1:|G|}$.

4.2 Path Probability

Updating the parameters to make the ground-truth path shorter does not guarantee that it will be shorter *relative to* the other paths. So we rewrite it as a properly normalized probability, taking into account all other candidates. The *path probability* of $x_{1:t}$ is defined by the following formula:

$$\mathrm{P}(X_{1:t} = x_{1:t} \in G_{1:t} \mid G_{1:t}) \triangleq \begin{cases} 1/|G_1| = 1 & \text{if } t = 1 \\ \prod_{u=1}^{t-1} \frac{\exp(-\boldsymbol{\delta}(x_u, x_{u+1}))}{Z_{G,u}} & \text{otherwise} \end{cases} \tag{4.1}$$

where

$$Z_{G,u} \triangleq \sum_{l \in G_u} \sum_{m \in G_{u+1}} P(X_u = l \mid G_{1:u}) \exp(-\boldsymbol{\delta}(l, m)),$$

see also appendix A.1 for the properness of this formula. The *node probability* $\mathrm{P}(X_t = x_t \mid G_{1:t})$, which appears in $Z_{G,t}$, is defined by marginalizing path probability as follows:

$$\begin{aligned}
\mathrm{P}(X_t = x_t \mid G_{1:t}) &= \sum_{x_{1:t} \in G_{1:t-1}\{x_u\}} \mathrm{P}(X_{1:t} = x_{1:t} \mid G_{1:t}) \\
&= \sum_{x_{1:t} \in G_{1:t-1}\{x_t\}} \prod_{s=1}^{t-1} \frac{\exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}} \\
&= \sum_{x_{1:t} \in G_{1:t-1}\{x_t\}} \left(\prod_{s=1}^{t-2} \frac{\exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}} \right) \times \frac{\exp(-\boldsymbol{\delta}(x_{t-1}, x_t))}{Z_{G,t-1}} \\
&= \sum_{x_{1:t} \in G_{1:t-1}\{x_t\}} \mathrm{P}(X_{1:t-1} = x_{1:t-1} \mid G_{1:t-1}) \times \frac{\exp(-\boldsymbol{\delta}(x_{t-1}, x_t))}{Z_{G,t-1}} \\
&= \sum_{x_{1:t-1} \in G_{1:t-1}} \mathrm{P}(X_{1:t-1} = x_{1:t-1} \mid G_{1:t-1}) \times \sum_{x_t \in \{x_t\}} \frac{\exp(-\boldsymbol{\delta}(x_{t-1}, x_t))}{Z_{G,t-1}} \\
&= \sum_{x_{t-1} \in G_{t-1}} \mathrm{P}(X_{t-1} = x_{t-1} \mid G_{1:t-1}) \times \frac{\exp(-\boldsymbol{\delta}(x_{t-1}, x_t))}{Z_{G,t-1}}.
\end{aligned}$$

As we can see, this has a recursive structure, and, by processing in a sequential manner from the start node, we can compute the node probability, and therefore

the path probability as well, with the time (and space) complexity linear to t since the size of each layer in G is limited. Then we can compute the probability for the whole interpretation path as $P(X_{1:|G|} = x_{1:|G|} \mid G_{1:|G|})$.

Equation (4.1) is designed to give higher values to the interpretation paths with shorter total distances (Theorem 1) and therefore the shortest path(s) should have the highest probability.

Theorem 1 (order accordance)**.** *In an interpretation graph G,* $\mathbf{len}(x_{1:t})$ *is smaller than* $\mathbf{len}(x'_{1:t})$ *if and only if* $\mathrm{P}(X_{1:t} = x_{1:t} \mid G_{1:t})$ *is greater than* $\mathrm{P}(X_{1:t} = x'_{1:t} \mid G_{1:t})$.

Proof. Given $\mathbf{len}(x_{1:t}) < \mathbf{len}(x'_{1:t})$, we have $\mathrm{P}(X_{1:t} = x_{1:t} \mid G_{1:t}) > \mathrm{P}(X_{1:t} = x'_{1:t} \mid G_{1:t})$ as below:

$$
\begin{aligned}
&\mathbf{len}(x_{1:t}) < \mathbf{len}(x'_{1:t}) \\
&\Leftrightarrow -\mathbf{len}(x_{1:t}) > -\mathbf{len}(x'_{1:t}) \\
&\Leftrightarrow \exp(-\mathbf{len}(x_{1:t})) > \exp(-\mathbf{len}(x'_{1:t})) \\
&\Leftrightarrow \exp(-\sum_{u=1}^{t-1} \boldsymbol{\delta}(x_u, x_{u+1})) > \exp(-\sum_{u=1}^{t-1} \boldsymbol{\delta}(x'_u, x'_{u+1})) \\
&\Leftrightarrow \prod_{u=1}^{t-1} \exp(-\boldsymbol{\delta}(x_u, x_{u+1})) > \prod_{u=1}^{t-1} \exp(-\boldsymbol{\delta}(x'_u, x'_{u+1})).
\end{aligned}
$$

Now, we divide both sides by the same (positive) value $\prod_{u=1}^{t-1} Z_{G,u}$:

$$
\begin{aligned}
&\Leftrightarrow \frac{\prod_{u=1}^{t-1} \exp(-\boldsymbol{\delta}(x_u, x_{u+1}))}{\prod_{u=1}^{t-1} Z_{G,u}} > \frac{\prod_{u=1}^{t-1} \exp(-\boldsymbol{\delta}(x'_u, x'_{u+1}))}{\prod_{u=1}^{t-1} Z_{G,u}} \\
&\Leftrightarrow \prod_{u=1}^{t-1} \frac{\exp(-\boldsymbol{\delta}(x_u, x_{u+1}))}{Z_{G,u}} > \prod_{u=1}^{t-1} \frac{\exp(-\boldsymbol{\delta}(x'_u, x'_{u+1}))}{Z_{G,u}}.
\end{aligned}
$$

So we have the objective inequality from Equation (4.1):

$$\Leftrightarrow \mathrm{P}(X_{1:t} = x_{1:t} \mid G_{1:t}) > \mathrm{P}(X_{1:t} = x'_{1:t} \mid G_{1:t}).$$

□

4.3 Learning Strategy

We define the loss function using cross entropy as follows:

$$
\begin{aligned}
\mathbf{loss}(x_{1:|G|}, x^*_{1:|G|} \mid G_{1:|G|}) \triangleq \sum_{x_{1:|G|} \in G_{1:|G|}} & -\mathrm{P}^*(x_{1:|G|} = x^*_{1:|G|}) \\
& \times \ln \mathrm{P}(X_{1:|G|} = x_{1:|G|} \mid G_{1:|G|}),
\end{aligned}
\tag{4.2}
$$

where P^* is an indicator function defined as:

$$\mathrm{P}^*(x_{1:|G|} = x^*_{1:|G|}) \triangleq \begin{cases} 1 \text{ if } x_{1:|G|} = x^*_{1:|G|} \\ 0 \text{ otherwise} \end{cases}$$

We can obtain the gradient by differentiating **loss** (Equation (4.2)) with respect to the parameters, and then apply stochastic gradient descent (SGD) to update the parameters to minimize the value of **loss**, which results in maximizing the path probability for the ground-truth path.

To recap, if we want to train a function, say, $\mathbf{m\_sym1}(x, y)$, we plug in this function for $\boldsymbol{\delta}$, that is, $\boldsymbol{\delta}(x, y) = \mathbf{m\_sym1}(x, y) = v[\mathbf{m}(x) + \mathbf{m}(y)]$. Although we can plug in any functions that can be constructed in the way explained in Section 3.2, the training process proposed in Section 4 can be applied only when the function is a learnable function, which is composed of the classifier and the parameter array. In the training process, specifically, the parameters are updated over multiple epochs by SGD so as to minimize the cross entropy of Equation (4.1) and the ground-truth path, that is, Equation (4.2), as described in Section 4.3. Note that this process updates the parameter array part in the function while not affecting the classifier part.

Using the resulting trained function $\boldsymbol{\delta}(x, y) = \mathbf{m\_sym1}(x, y)$, we can evaluate the performance of this function by measuring its prediction accuracy on test data. It is expected that the final performance will fluctuate depending on which function is set for $\boldsymbol{\delta}$. Assuming that the training process was carried out properly, this fluctuation reflects the difference in the performance of the classifier. Here, the classifier is a combination of the basic features introduced in Section 3.1. Therefore, by comparing and evaluating these, it is possible to measure what kind of combination of features is effective.

4.4 Accuracy

With the learned parameters, we can go back to the original method [30] to find the most plausible path. Then we evaluate our model based on how accurately it can predict each chord interpretation by specifying the shortest path(s) in the interpretation graph. If there is more than one shortest path, which is not uncommon when the model is relatively simple, we compute a weighted average for each node in proportion to how many paths go through the node as in Figure 4.1.

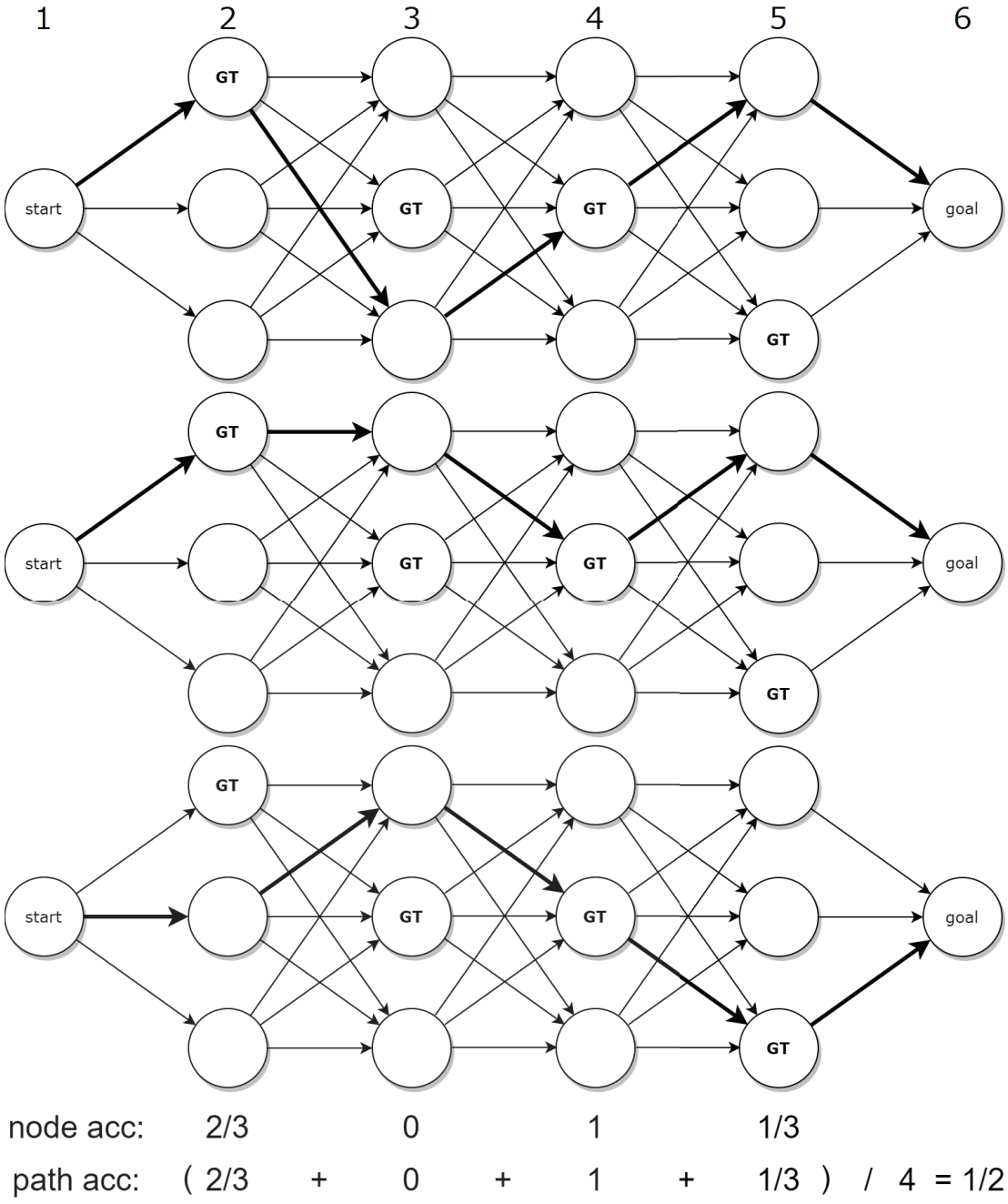

Figure 4.1: Sample calculation of node and path accuracies.

As can be seen from the figure, the path accuracy is calculated as the average of node accuracies. Note that, if we look at the last three layers, i.e., layers 4 to 6, there are just two unique paths. But the accuracy of the fourth layer is 1/3 instead of 1/2. We provide an efficient algorithm to compute the path accuracy in

appendix A.3.

Finally, we illustrate how the paths are selected with some distance models with concrete values in Figure 4.2. In this figure, Although this example uses a very simple graph, the distance models are also very simple and can not reduce the shortest path to a single path; **m_dest** with parameters $u = [2.0, 1.0]$ yields C : I $\rightarrow$ IV $\rightarrow$ V $\rightarrow$ I and a : III $\rightarrow$ C : IV $\rightarrow$ V $\rightarrow$ I as the shortest paths resulting in an accuracy of $(1/2 + 1 + 1 + 1)/4 = 87.5\%$. On the other hand, **m_sym2** with parameters $v = [1.0, 2.0]$ yields C : I $\rightarrow$ IV $\rightarrow$ V $\rightarrow$ I and a : III $\rightarrow$ VI $\rightarrow$ VII $\rightarrow$ III as the shortest paths, and this results in an accuracy of $(1/2+1/2+1/2+1/2)/4 = 50.0\%$.

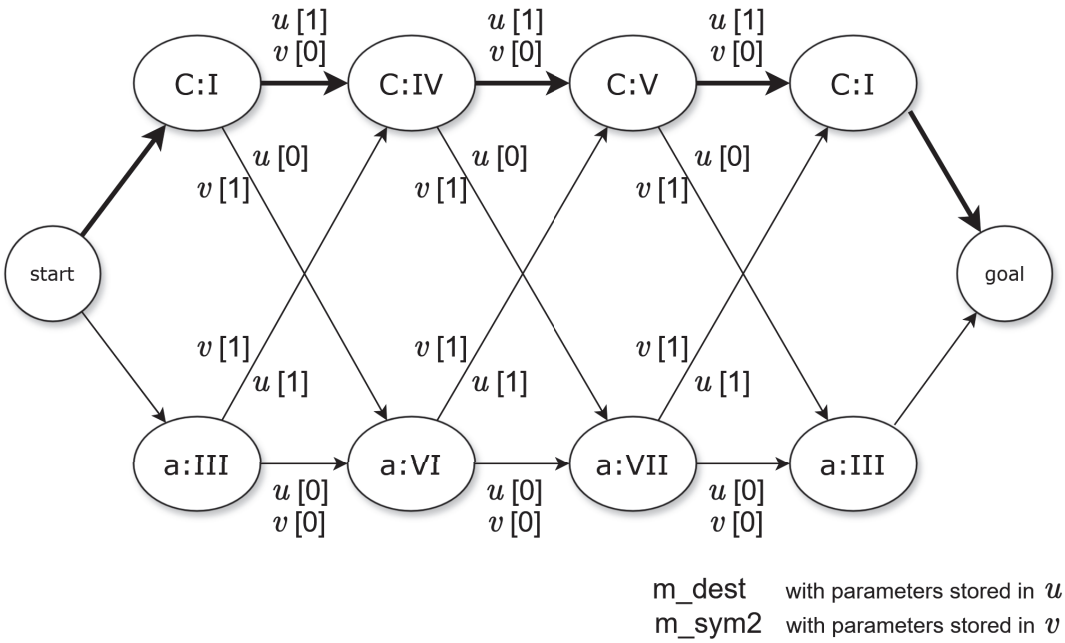

Figure 4.2: Simplified interpretation graph, in which every chord has just two interpretations. The ground-truth path is indicated by thick arrows.

Chapter 5

Comparative Analysis

5.1 Experimental Setup

In our experiments, we use the dataset published by Gotham et al. [11, 12] which consists of human-annotated pieces of classical music spanning from the Baroque period to the late 19th century. Analyses are provided in `rntxt` format by Tymoczko, Gotham, Cuthbert, & Ariza [38], while the musical scores are also available in .mxl or other formats. There are 405 pieces (2,992 phrases, 80,414 chords), and we regard every phrase as a unit (i.e., to which we predict the interpretation sequences); however, when a phrase exceeds 50 chords, we divide it into units each of which does not exceed 50 chords (e.g., in the case of a phrase consisting of 130 chords, we simply divide it into three parts: 50, 50, and 30 chords). Then we use 80% for training, 10% for validation, and remaining 10% for the test.

We extracted key, degree and secondary chord (applied chord) information from `rntxt` using `music21` library by Cuthbert & Ariza [7]. We expand secondary chords in three ways. In *original*, every secondary chord is converted into the prevailing key with the degree computed by modulo operator (e.g., chord sequence C : IV/V $\rightarrow$ V/V $\rightarrow$ V $\rightarrow$ I is converted into C : I $\rightarrow$ II $\rightarrow$ V $\rightarrow$ I). In *local*, every secondary chord is interpreted in local key (e.g., C : IV/V $\rightarrow$ V/V $\rightarrow$ V $\rightarrow$ I is converted into G : IV $\rightarrow$ V $\rightarrow$ C : V $\rightarrow$ I). In *epsilon*, a tonic chord is added at the end of every local key section to express pivot chord modulation or 'ϵ-transition' proposed by [42] (e.g., C : IV/V $\rightarrow$ V/V $\rightarrow$ V $\rightarrow$ I is converted into G : IV $\rightarrow$ V $\rightarrow$ I $\rightarrow$ C : V $\rightarrow$ I). In this experiment, we examine the effects of these differences as well. After these expansions, we omit all repetitions of the same chord interpretation (e.g., V $\rightarrow$ I $\rightarrow$ I $\rightarrow$ I $\rightarrow$ I $\rightarrow$ IV is converted into V $\rightarrow$ I $\rightarrow$ IV).

We initialize all parameter values to zero and train them by mini-batch SGD with batch size=100 and learning rate=0.001. We continue training until there is

no change in accuracy over the validation set for two consecutive epochs then pick the parameter which gives the highest validation accuracy.

5.2 Evaluation Targets

For a pair of chord interpretations, there are six basic features. In this experiment, we compare all combinations of them in which the same feature is used at most once, and all of these combinations can be expressed using elemental and compound functions. If we use at most one elemental function from each of Sections 3.2.1 to 3.2.3, we can construct 623 functions by the calculation similar to [44][1]. In addition to these, elemental functions of type (i) can be used at the same time (e.g., **m_src** and **m_dest** do not overlap in the use of basic features). Considering this, the overall number becomes 1,540 (see appendix B.1 for details). We evaluate these functions in three settings (i.e., *original*, *local*, and *epsilon*).

[1]We treat the region as a variant of the tonic feature, therefore, we do not count the occurrence of m in r (Equation (3.9)) as a use of mode feature.

5.3 Results and Discussions

5.3.1 Overview

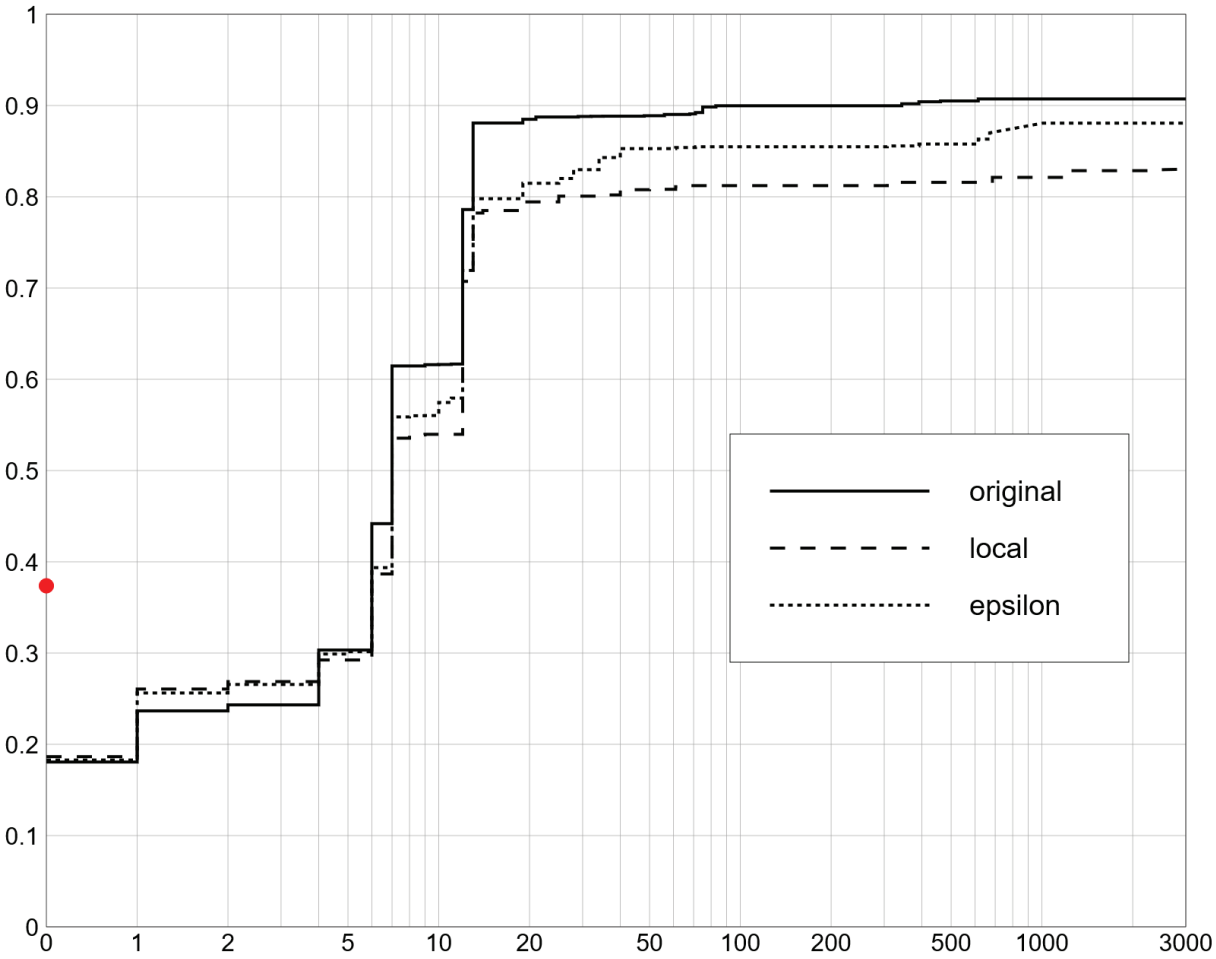

Figure 5.1: Best accuracy **y** with at most **x** #EP. Solid, dotted, and dashed lines represent the performances of *original*, *local*, and *epsilon* settings, respectively. Red dot is the score of **tps** from Equation (2.4). It is placed on the left end of the chart because **tps** does not have any learnable parameters.

Figure 5.1 shows the relationships between #EP and the maximum accuracy that could be achieved within the #EP. Despite the differences in settings, the shape of each graph is roughly the same. The leftmost area with around 20-30% accuracy is where the #EP is so small that only the simplest elemental functions like **m_src**, and **d_min_steps** can be used. Next, the accuracies leap to over 70% at 12 #EP, which is when *addition* of **min_steps** of tonic feature and one side functions of

degree feature become possible (**t_min_steps** + **d_src** achieved 78.60% in *original*, **r_min_steps**+**d_dest** achieved 70.73% in *local*, and **t_min_steps**+**d_dest** achieved 71.03% in *epsilon*).

At 13 #EP, the scores then surge again. 13 is the minimum #EP value where elemental functions of all three features can be combined (**m_sym2**+**r_min_steps**+ **d_dest** achieved 88.08% in *original*, the same compound function achieved 78.22% in *local*, and **m_sym2** + **t_min_steps** + **d_dest** achieved 79.79% in *epsilon*)[2]. From the fact that the scores almost stop increasing there, we believe that it is crucial to use all three features, and the combination of them is not necessarily by *multiplication*. In *epsilon*, however, the score still increases until 40 #EP though relatively slowly (for example, **m_sym2** × **r_min_steps** + **d_dest** (19 #EP) achieved 81.48%, **m_sym2** + **r_min_steps** + **d_sym2** (28 #EP) 82.01% ,and **m_sym2** × **t_min_steps** + **d_sym1** (40 #EP) 85.28%). All of them have the operation of *multiplication*. Note that **d_sym1** can be thought of a simplified version of **d_src** × **d_dest**. Also note that **m_sym1** and **m_sym2** contain *multiplication* in this sense, and are preferred to **m_src** and **m_dest** because they have almost the same #EP); it seems a little more complication, mainly by introducing *multiplication*, over 13 #EP actually works effectively in *epsilon*.

The maximum accuracies are 90.72% in *original* (by **m_src**×**m_dest**×**r_min_steps**× **d_sym2** ($2 \times 2 \times 7 \times 22 - 1 = 615$ #EP)), 82.95% in *local* (by **m_src** × **m_dest** × **r_oneway_steps**×**d_src**×**d_dest** ($2 \times 2 \times 12 \times 7 \times 7 - 1 = 2,351$ #EP)), and 88.07% in *epsilon* (by **m_sym1** × **t_oneway_steps** × **d_sym1** ($3 \times 12 \times 28 - 1 = 1,007$ #EP)). As just described, considering a secondary chord as belonging to the temporal key (i.e., *local*) is more difficult than ignoring the key difference (i.e., *original*). However, supplementing an additional chord that works as a pivot (i.e., *epsilon*) helps to analyze this phenomenon.

5.3.2 Comparison of Elemental Functions

[2]We exclude **d_min_steps** here because, as described later, interval scales of degree feature have almost no effect.

Table 5.1: Performance of elemental functions and some other combined ones. The column **alone** is the accuracy gain (from 18.28% where all edges have 0 distance) when used alone. The columns **addition** and **multiplication** are the average accuracy gains when combined with the other (learnable) functions by *addition* and *multiplication* respectively. **add/#EP** and **mult/#EP** are each the values of **addition** and **multiplication** divided by the average increases of #EP. Some compound functions are included for comparison with indicator '†'.

elemental function	**#params**	**alone**	**addition**	**add/#EP**	**multiplication**	**mult/#EP**
m_src	2	7.35%	2.10%	2.10%	3.51%	0.0431%
m_dest	2	7.22%	1.90%	1.90%	3.70%	0.0390%
m_src × **m_dest** (†)	4	8.28%	10.56%	3.52%	11.55%	0.0234%
m_sym1	3	8.28%	10.49%	5.25%	11.31%	0.0522%
m_sym2	2	0.00%	6.09%	6.09%	7.31%	0.1214%
t_oneway_steps	12	11.24%	31.18%	1.22%	32.56%	0.0089%
t_min_steps	7	11.31%	31.20%	1.39%	32.16%	0.0102%
r_oneway_steps	12	19.82%	31.49%	0.59%	32.65%	0.0012%
r_min_steps	7	19.82%	31.48%	1.00%	32.41%	0.0036%
d_src	7	20.94%	7.30%	1.28%	8.19%	0.0060%
d_dest	7	21.08%	8.35%	0.36%	9.33%	0.0060%
d_oneway_steps	7	0.85%	2.18%	0.62%	1.32%	0.0206%
d_min_steps	4	-0.27%	1.87%	2.83%	1.13%	0.0159%
d_src × **d_dest** (†)	49	25.02%	28.23%	5.20%	29.26%	0.0536%
d_sym1	28	23.88%	26.93%	2.86%	27.76%	0.0161%
d_sym2	22	23.78%	26.86%	5.25%	27.65%	0.0541%
average		13.04%	16.14%	2.59%	16.99%	0.0297%

Next, we computed how much each of the elemental functions in Section 3.2 could increase the accuracy from nothing (**alone**) and on average when combined by *addition* (**addition**) or by *multiplication* (**multiplication**) in *epsilon* setting (from now on, we mainly use the result from *epsilon* setting). Table 5.1 shows the results. Elemental functions except type(i) performed better when combined with others than used alone. Presumably, this is because elemental functions have a kind of synergy effect when combined with other elemental functions. Especially, **m_sym2** contributes only when it is combined with others.

Elemental functions of type (i), despite its very simple structure, had higher gains when used alone than when used in *addition* or *multiplication*; these type (i) functions seem to pick up significant information when used in **alone**, however, these functions intrinsically correlate with other functions and thus each impact becomes relatively smaller when combined. In general, they have the advantage of being simple and parsimonious. For the type (i) elemental functions for degree features, **d_src** and **d_dest**, are beneficial to construct functions of low #EP. In fact, **m_sym2**×**r_min_steps**+**d_src**+**d_dest** was the smallest model that achieved over 80% in all three settings. This type of combination was not considered in [44]. For mode features, however, this advantage is not so significant because of **m_sym2**, which looks at both sides with the same amount of parameters.

Regarding *addition* and *multiplication*, the latter scored better in almost all cases. The only exception is the stepwise distances (i.e., type (ii)) for the degree feature, which performed very poorly,[3] but this will be discussed in more detail later. Even though *multiplication* performed better in many cases, it also increases the number of necessary parameters (and consequently #EP), disproportionately to the accuracy gain. Columns **add/#EP** and **mult/#EP** in Table 5.1 show the average accuracy gains per #EP, revealing the difference of about two orders of magnitude between them. Since our distance functions are to help understand and analyze the harmonic phenomena of tonal music, it is not desirable to overcomplicate them to acquire small gains. Therefore, *addition* is, we believe, preferable to *multiplication* in many cases (except, maybe, when #EP is 40 or less in epsilon setting, and type (ii) functions for the degree feature). In this regard, the additive structure of the original TPS (Equation (2.4)) is, from our experiments, indeed reasonable.

To see if it is worth distinguishing the direction of transitions, we can compare directional functions to the corresponding non-directional (or symmetric) functions (i.e., we can compare ***_src**×***_dest** to ***_sym1** or ***_sym2**, and ***_oneway_steps** to ***_min_steps** with the values in Table 5.1). **m_src** × **m_dest** performed only

[3]Having said that, from the nature of our framework, it is not likely that a function has a negative effect in a general way. So the negative gain of **d_min_steps** should be due to an accidental effect of the training process and the selection of the test set.

slightly better than **m_sym1**, but clearly better than **m_sym2**. Therefore, regarding the mode feature, it was meaningful to distinguish major or minor even when they continue, but directions were not worth considering. There are two such pairs in the tonic feature but neither of them differed much. Only for the elemental functions for the degree feature, could we observe over 1% gain. However, **d_src** $\times$ **d_dest** needs more than twice as many parameters as **d_sym2** to improve the accuracy by about 1.5%, and stepwise distances of degree were not so effective in the first place. From the above, it was found that taking direction into account does not lead to much improvement in performance, while heavily increasing #EP; this fact also supports the structure of TPS which is designed carefully to keep symmetricity.

For the tonic feature, we tried the stepwise distances of region in addition to tonic itself, because original TPS utilizes region, not tonic, to define the distance. And indeed, region steps worked far better than tonic steps. However, the difference disappeared when combined with others. This is because the difference is caused by Equation (3.9), a very simple conversion, and this gap is easily covered when combined with other features.

Table 5.2: Accuracy gains of non-learnable functions from TPS.

non-learnable function	**alone**
tps_region	18.98%
tps_chord	0.19%
tps_basicspace	19.06%
tps (Equation (2.4))	19.09%

We also checked the performances of the original TPS (as non-learnable functions explained in Section 3.2.4). Although **tps_chord** could not improve the score well, the other functions worked equally well. And **tps_basicspace**'s score is almost the same as **tps_region**'s because the former behaves almost the same as the latter in this task. As a result, the score of **tps** (i.e., the addition of these three functions) is not much different from **tps_region** and **tps_basicspace**. And as **r_min_steps** (when used alone) can be said to be a learnable version of **tps_region**, it performed better than **tps_region** (and **tps** as well) even though it has only 6 #EP. As for **tps_chord**, its low performance corresponds to those of **d_oneway_steps** and **d_min_steps**.

Table 5.3: Accuracy gains from 19.82% of **m_sym2** × **r_min_steps** by the degree features and **tps_chord**.

composit function	acc gain	#EP
m_sym2 × **r_min_steps** + **tps_chord**	0.33%	13
m_sym2 × **r_min_steps** + **d_min_steps**	2.90%	16
m_sym2 × **r_min_steps** + **d_oneway_steps**	3.43%	19
m_sym2 × **r_min_steps** × **d_min_steps**	21.82%	55
m_sym2 × **r_min_steps** × **d_oneway_steps**	21.66%	97
m_sym2 × **r_min_steps** + **d_src**	42.14%	19
m_sym2 × **r_min_steps** + **d_dest**	43.38%	19

tps_chord is a stepwise distance function of the degree feature, so the type (ii) functions, **d_min_steps** and **d_oneway_steps**, are basically the learnable versions of **tps_chord**. In fact, all of these perform very poorly, as mentioned above. Table 5.3 shows accuracy gains of them when combined with **m_sym2** × **r_min_steps**, which is the learnable counterpart of **tps_region** (and also **tps_basicspace** in this task). When combined using *addition*, their performances were low. However, switching to *multiplication* significantly improved the performance. This means that when using the type (ii) functions of the degree feature, it should not be considered independently of the key changes. Whereas *addition* is a method of combining two arguments independently. In that sense, the additive structure of TPS is inappropriate for type (ii) functions of the degree feature. Table 5.3 also includes combinations of the degree's type(i) functions for comparison. They work quite well with *addition* and are also very economical in terms of #EP. After all, type (ii) functions don't seem particularly easy to use for the degree feature.

5.3.3 Comparisons with Other Models

Here, taking **m_sym2** × **r_min_steps** + **d_sym2** as an example, we look at the parameters and operation results. We take this model because its structure is convenient to compare with that of TPS while achieving over 80% accuracy in all three settings[4] with $2 \times 7 + 22 - 2 = 34$ #EP.

Learned parameters

The learned parameters of the model are shown in Table 5.4. The parameters for the **m_sym2** × **r_min_steps** part are shown in (a), and for the **d_sym2** part are shown in (b).

[4] 88.29%, 80.20%, and 83.72% in *original*, *local*, and *epsilon* settings, respectively.

Table 5.4: Learned parameters (learned in *epsilon* setting) of **m_sym2** × **r_min_steps** + **d_sym2** (34 EPs). **c1**, **r1**, **c2**, and **r2** are **r_min_steps**(x, y), **m_sym2**(x, y), y^{degree}, and x^{degree} respectively. As in Figure 3.4, cells that simply duplicate the values of other cells are enclosed in parentheses. The values adjusted to fall within the range of 0 to 1.

(a)	**(c1=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r1=)0	0.0000	0.8507	0.8793	0.8630	0.8446	0.3795	0.8205
1	0.5402	1.0000	0.7646	0.7874	0.5387	0.5559	0.7885

(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r2=)0	0.5548	0.1455	0.3339	0.1870	0.0000	0.2144	0.0732
1	(0.1455)	(0.5548)	0.3971	0.3228	0.1686	0.3942	0.2850
2	(0.3339)	(0.3971)	(0.5548)	0.3198	0.3357	0.4408	0.4182
3	(0.1870)	(0.3228)	(0.3198)	(0.5548)	0.1871	0.3573	0.3043
4	(0.0000)	(0.1686)	(0.3357)	(0.1871)	(0.5548)	0.2088	0.2149
5	(0.2144)	(0.3942)	(0.4408)	(0.3573)	(0.2088)	(0.5548)	0.2909
6	(0.0732)	(0.2850)	(0.4182)	(0.3043)	(0.2149)	(0.2909)	(0.5548)
(sum)	1.5088	2.2680	2.8003	2.2331	1.6699	2.4612	2.1413

Prediction example

Figure 5.2 shows two examples of predictions of our model. (a) has a modulation, and our model correctly predicted all but the exact moment when the modulation occurred. In (b), there is a secondary chord, so we show the predictions in three settings; prediction in *original* in this example only needs chord interpretations in A♭ key and the model succeeded in predicting all of them, prediction in *local* must have a different key at the secondary chord and the model predicted it correctly, however, it failed to return to the original key after that, in *epsilon*, on the other hand, adds E♭:I after the secondary chord, and the model predicted the whole interpretation successfully.

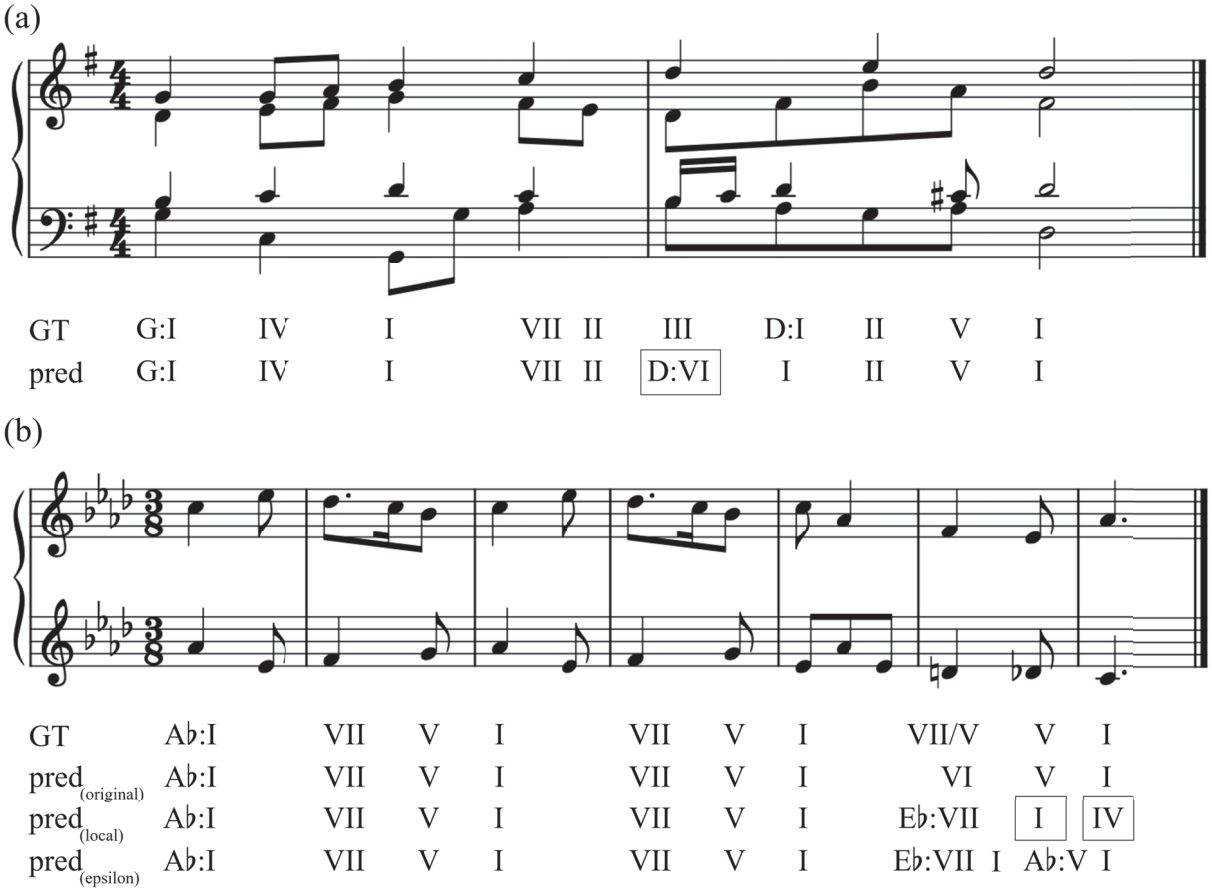

Figure 5.2: Examples of annotated chord interpretations (GT) and predictions (pred). Chord interpretations that the model failed to predict correctly are written in boxed digits. (a) An example with a key modulation. (b) An example with an secondary chord.

Comparison in key

The **m_sym2** $\times$ **r_min_steps** part can be compared with **region** in Equation (2.1) as in Table 5.5 which contains the results of **region** in each cell of Table 5.4 (a). As for the values, there is not much noticeable similarity other than the relatively small values when **c1** $= 0$ or 5 (i.e., the distances of I $\leftrightarrow$ I, ii, iii, IV, V, vi and i $\leftrightarrow$ i, ♭III, iv, v, ♭VI, ♭VII are small). However, our model distinguishes **r1** (i.e., whether mode switches) but **region** does not. This distinction causes approximately 8% gain with 7 more #EPs compared to the model without it (i.e., **r_min_steps** + **d_sym2** $(7 + 22 - 2 = 27$ #EP), which achieved 80.31% for *original*, 72.86% for *local*, and 75.67% for *epsilon*). We therefore consider this a meaningful distinction. In fact, **tps** from Equation (2.4) is designed so that transitions to keys other

than related keys (Equation (2.2)) have larger values through more complicated calculations (Equation (2.3)).

Table 5.5: Distance values of **region** in Equation (2.1) for each case of **c1** = **r_min_steps**(x, y) and **r1** = **m_sym2**(x, y).

	(c1=)0	**1**	**2**	**3**	**4**	**5**	**6**
(r1=)0,1	0	5	2	3	4	1	6

Figure 5.3 (a-d) shows the distances between keys by our model (Table 5.4 (a)) and **tps** (Equation (2.4)) in the "regional space" format, which is a combination of the fifth cycle (vertical axis) and the parallel/ relative major/ minor cycle (horizontal axis) proposed in [40] and adopted in [20].[5] The proximity relationships described above are also valid here. But, in our model, the area around the center (roughly the area of related keys) is somewhat clearly separated from the area outside it, but in **tps**, the values increase monotonically both vertically and horizontally from the center, and the boundary line is ambiguous. In our model, the values seem to generally fall into four areas (i.e., 0, 0.3795, around 0.55, and over 0.7), rather than chopping the values as finely as **tps**. In this study, we have defined the way to use the basic features based on the three types listed in Section 3.2, but from these results, if theoretical simplicity is sacrificed, the number of parameters could be further reduced without significant performance loss by, for example, ignoring the distinction between **c1** = 2, 3, and 6.

Another difference is that although both models generally assign small values to the area of related keys, our model includes I $\leftrightarrow$ ♯i, iv and i $\leftrightarrow$ ♭I, V instead of the parallel keys (i.e., I $\leftrightarrow$ i). The transitions I $\leftrightarrow$ iv and i $\leftrightarrow$ V are the intervals of the perfect fifth, accompanying major/ minor shifts, and are used as often as or more than I $\leftrightarrow$ ii, ii, vi and i $\leftrightarrow$ ♭III, ♭VI, ♭VII. Typical usage is when transposing from a major key tonic to a minor key with a dominant motion (this is I $\rightarrow$ iv and i $\leftarrow$ V), or when the secondary dominant is used in a minor key.[6] The model assigns I $\leftrightarrow$ ♯i and i $\leftrightarrow$ ♭I small values though they appear rarely; this is because **r_min_steps** ignores the direction of regional transition, so they are placed in the same block as I $\leftrightarrow$ iv and i $\leftrightarrow$ V. This problem can be avoided by using **r_oneway_steps** instead, but this increases the number of parameters while, as can be seen from Figure 5.1, it does not improve performance much.

It should be noted that our model assigns large values to I $\leftrightarrow$ v and i $\leftrightarrow$ IV even though these are also transitions with perfect fifth intervals (in fact, **tps** gives the same values for them). But in case of **r_min_steps**, the number of steps for

[5] We have also included the same diagram in Tonnetz style (see Appendix B.2).

[6] In *epsilon* setting, the progression c : I $\rightarrow$ V/V causes the modulation c $\rightarrow$ G $\rightarrow$ c.

I $\leftrightarrow$ v is 2 and I $\leftrightarrow$ iv is 4.And indeed, the typical usage mentioned in the previous paragraph can not be applied here.

As for the parallel keys, it seems to be too large even though it is combined with less frequent transitions I $\leftrightarrow \sharp$iv and i $\leftrightarrow \flat$V by **r_min_steps**. This is presumably compensated for by the + **d_sym2** part. In other words, in the case of the parallel keys, the value of **d_sym2** is likely to be smaller because the degree transitions are expected to be similar to when it happened within a key, and as a result, the **m_sym2** $\times$ **r_min_steps** part would not have needed to be very small. Conversely, I $\leftrightarrow \sharp$iv and i $\leftrightarrow \flat$V are likely to be given relatively large values by **d_sym2**, and as a result, the **m_sym2** $\times$ **r_min_steps** part would not have needed to be very large.

In addition, we also include the distances based on Chew's Spiral Array model [5][7] in Figure 5.3 (e, f). As with **tps**, the values are also gradually increasing at about the same rate, rather than being separated clearly. But while our model and **tps** generally assign lower values to related keys, the Spiral Array is much concerned with the key tonics than with major/ minor differences, so it does not correspond well to the area of related keys. Also, contrary to our model, I $\leftrightarrow$ v and i $\leftrightarrow$ IV are given smaller values than I $\leftrightarrow$ iv and i $\leftrightarrow$ V.

Comparison in degree

The **d_sym2** part can be compared with **chord** in Equation (2.1) as in Table 5.6 which contains the results of **chord** in each cell of Table 5.4 (b). Where values are duplicated in Table 5.4 (b) (i.e., cells with parentheses), they are also the same in Table 5.6, which is a natural result since **d_sym2** is designed after **chord** in the first place. The diagonal component of Table 5.4 (b) is maximum, while in Table 5.6 it is minimum, this is because our approach does not deal with duration and this is one of the limitations of our current approach.

As for other values, $(\mathbf{r2}, \mathbf{c2}) = (0, 4), (0, 6)$ (and their transpositions) have particularly small values in Table 5.4 (b). These represent the distances of I $\leftrightarrow$ V and I $\leftrightarrow$ VII.[8] Regarding the second smallest group, we can find $(\mathbf{r2}, \mathbf{c2}) = (0, 1), (0, 3), (1, 4), (3, 4)$ (and their transpositions) fall within the range of [0.1, 0.2). These represent the distances of I $\leftrightarrow$ II, I $\leftrightarrow$ IV, II $\leftrightarrow$ V, and IV $\leftrightarrow$ V. If we roughly consider I, III, VI are tonic (T), V, VII are dominant (D), and II, IV are subdominant (SD), all relationships in the smallest group represent the relationship

[7]The algorithm of [5] does not use the distance between keys. However, since the coordinates of keys are defined there, we use them to calculate the distance as the Euclidean distance between them. We adopted the configuration in [5], i.e., $w = u = \omega = v = (0.516, 0.315, 0.168)$, $h = \sqrt{2/15}$, $r = 1$, $\alpha = 1$, and $\beta = 0$.

[8]Note that the indices of **r2** and **c2** are degree minus 1. For example, the cell $(\mathbf{r2}, \mathbf{c2}) = (0, 4)$ represents the relationship between degrees I and V.

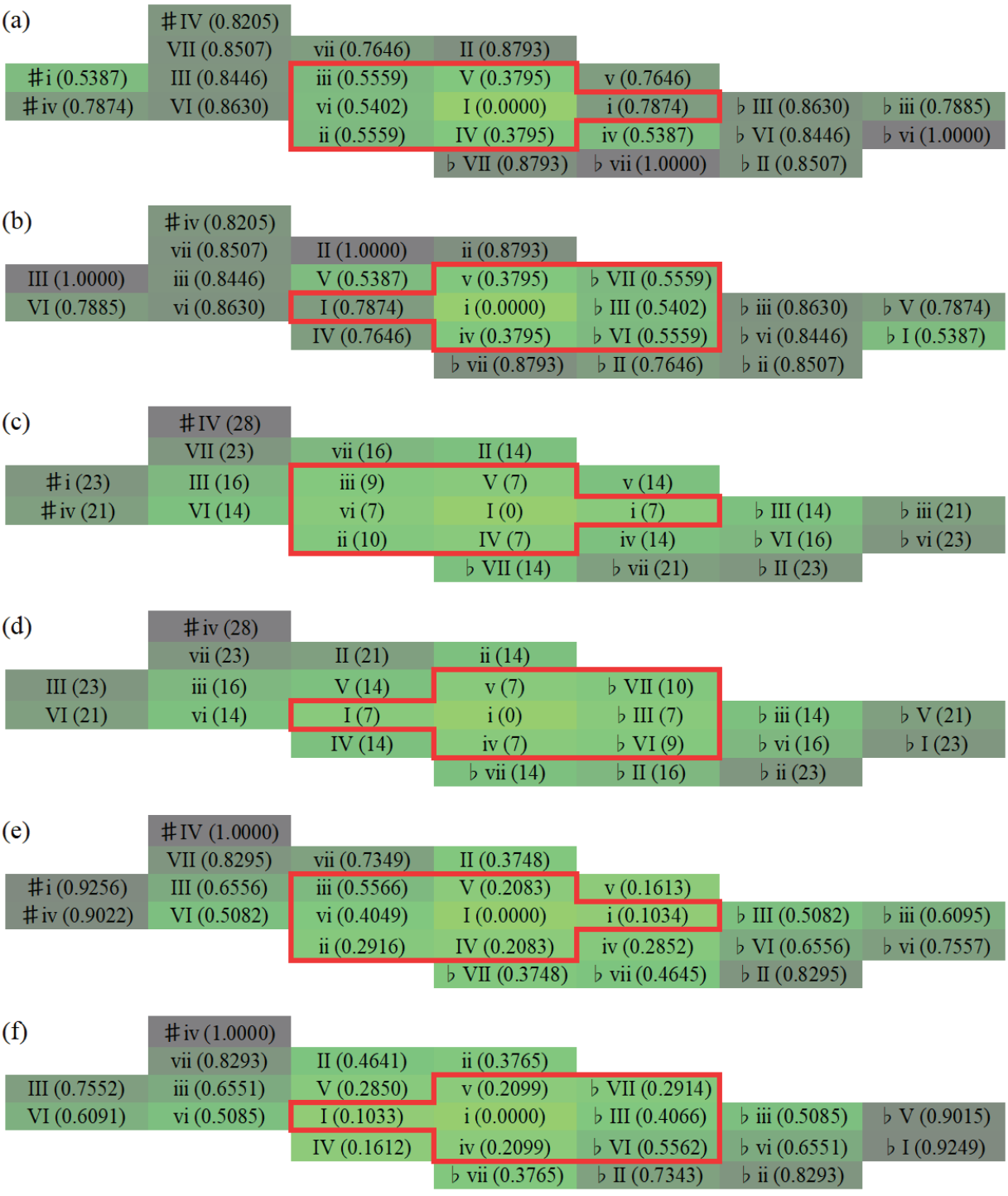

Figure 5.3: Distances between each key when degrees are fixed to I. The keys surrounded by frames are related keys (Equation (2.2)). (a) Distance by our model (Table 5.4 (a)) from major key, (b) from minor key. (c) Distance by **tps** (Equation (2.4)) from major key, (d) from minor key, and (e) distance by Spiral Array model by Chew [5] from major key, (f) from minor key. See appendix B.2 for a Tonnetz representation.

of T↔D, and those of the second smallest group represent the relationships of T↔SD and SD↔D. This is somewhat consistent with our musical knowledge.

If we compare it with Table 5.6, $(\mathbf{r2}, \mathbf{c2}) = (0,3), (0,4), (1,4)$ (and their transpositions) have small values and $(\mathbf{r2}, \mathbf{c2}) = (0,2), (1,3), (2,4), (3,5)$ (and their transpositions) have large values just like Table 5.4 (b), but there is not much similarity in other areas. In fact, among the elemental functions, **d_min_steps** is the closest in concept to **chord**. However, as mentioned above, neither performance was good. These functions consider only the number of steps. However, Table 5.4 (b) shows, for example, that I ↔ V is much smaller than II ↔ VI or III ↔ VII for the same 4-step moves. Therefore, it seems that nominal scale, which takes into account the absolute position of the move, is more effective for degree moves than simply looking at the number of steps.

Table 5.6: Distance values of **chord** in Equation (2.1) for each case of $\mathbf{c2} = y^{\text{degree}}$ and $\mathbf{r2} = x^{\text{degree}}$.

	(c2=)0	**1**	**2**	**3**	**4**	**5**	**6**
(r2=)0	0	2	3	1	1	3	2
1	2	0	2	3	1	1	3
2	3	2	0	2	3	1	1
3	1	3	2	0	2	3	1
4	1	1	3	2	0	2	3
5	3	1	1	3	2	0	2
6	2	3	1	1	3	2	0
(sum)	12	12	12	12	12	12	12

Table 5.7: Distance values from [17], [2] for each case of $\mathbf{c2} = y^{\text{degree}}$ and $\mathbf{r2} = x^{\text{degree}}$.

	(c2=)0	**1**	**2**	**3**	**4**	**5**	**6**
(r2=)0	-	0.1961	0.2092	0.1692	0.1684	0.1901	0.2188
1	0.1757	-	0.2500	0.2101	0.1639	0.2012	0.1946
2	0.1859	0.2237	-	0.2160	0.1988	0.2174	0.2237
3	0.1684	0.2000	0.2370	-	0.1667	0.2299	0.2088
4	0.1616	0.2088	0.2237	0.1815	-	0.1927	0.2062
5	0.1984	0.1838	0.2119	0.1972	0.1799	-	0.2222
6	0.1709	0.2404	0.2404	0.2208	0.1938	0.2387	-
(sum)	1.0609	1.2528	1.3722	1.1948	1.0715	1.2700	1.2743

Table 5.7 shows the data obtained from the experiments of [2][9]. Here, (0, 3), (0, 4), (1, 4), (3, 0), (3, 4), (4, 0), and (6, 0) have small values, which are generally consistent with the smallest and the second smallest groups in Table 5.4 (b). It is also in common that I and V have particularly small values when averaged over columns. In fact, the column averages except VII are ordered as I $<$ V $<$ IV $<$ II $<$ VI $<$ III, and this is also consistent. As for VII, since Table 5.7 is obtained from an experiment only on the major key context, it presumably reflects the effect of the diminished chord, which is in itself more dissonant than other diatonic triads, more strongly.

[9]The contents are from Table 8.2 of [17] and we take the reciprocal following [27]. Note that it distinguishes directions, and that the experiment is for major keys only.

Chapter 6

Further Extension: Encompassing Basic Space

6.1 Introduction

In this chapter, we aim to obtain optimal distances between pitch classes (PCs) and chords, through the task of finding the most plausible path in chord interpretation, from chroma vectors. First, we introduce some distance models which can calculate the distance between a chroma vector and a chord interpretation (Section 6.2). Second, we explain the way how to embed the models of Section 6.2 into the method of Sections 3 and 4 to enable the method to receive chroma vectors instead of chord symbols (Section 6.3). Finally, we conduct an experiment and present the results (Section 6.4).

6.2 Between Chroma Vectors and Chord Interpretations

In this section, we introduce chroma distance models which are inspired by the structure of basic space (and the basic space itself is one of them).

A chroma vector is a 12-dimensional vector, the element of which represents its membership (1/0) or graded salience of the corresponding PC. We define the distance between a chroma vector and a chord interpretation as the sum of all distances between PCs and the chord interpretation.

Firstly, we can calculate this distance using basic space. For example, the distance between the chroma vector [1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0] and the chord interpretation C:I can be calculated as the inner product of the chroma vector and the vector generated from the basic space as in Figure 6.1 (as the number of gray

boxes).

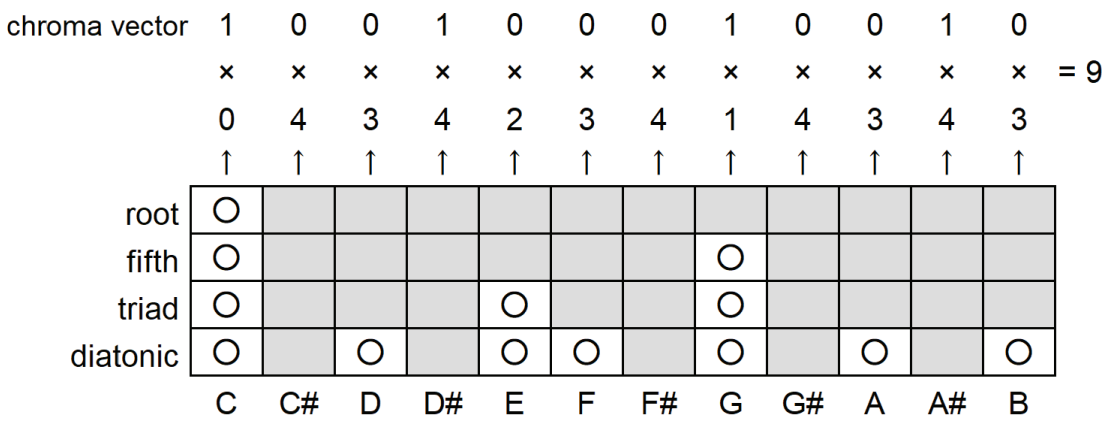

Figure 6.1: The distance between [1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0] and C:I based on the basic space.

This means basic space divides PCs into five categories, namely, *root* (i.e., C in this case), *triad* (i.e., E), *fifth* (i.e., G), *diatonic* (i.e., D, F, A, and B), and the others then gives the predefined PC-level distance values as in Table 6.1.

root	**third**	**fifth**	**diatonic**	**other**
0	2	1	3	4

Table 6.1: The PC-level distance values from basic space.

But, how to classify PCs and what distance values to apply are not obvious. So we try other possible models. Although the distance values are predefined in the original basic space, the distance values in the following models will be learned by machine learning given an annotated dataset.

	PC classification	**params**
basicspace	root/third/fifth/diatonic/other	0
ch_dist_0	-	0
ch_dist_2	chord/other	2
ch_dist_3	chord/diatonic/other	3
ch_dist_5	root/third/fifth/diatonic/other	5
ch_dist_10	(root/third/fifth/diatonic/other) ×(major/minor)	10

Table 6.2: Chroma distance models. **params** is the number of learnable parameters.

The first model, **ch_dist_2**, simply considers if the PC is in the chord note (i.e., *root*, *triad*, and *fifth*) or not. The next model, **ch_dist_3**, distinguishes whether the PC is diatonic or not in addition to the distinction by chord membership. The next one, **ch_dist_5**, uses the same categories as those of basic space. And finally, **ch_dist_10** uses the same five categories but also distinguishes scales (major or minor). We also define a dummy model, **ch_dist_0**, for comparison. This one always returns 0 regardless of what input is given. Table 6.2 shows the chroma distance models defined above.

	C:I	**e:i**	**C:vi**	**d:VII**	**G:I**
basicspace (Table 6.1)	6	6	9	7	8
ch_dist_0	0.0000	0.0000	0.0000	0.0000	0.0000
ch_dist_2 (Table 6.5)	2.4576	2.4576	4.9152	2.4576	4.9152
ch_dist_3 (Table 6.6)	2.0414	2.0414	4.0828	2.6578	4.0828
ch_dist_5 (Table 6.7)	6.8702	6.8702	9.2163	7.9469	10.0688
ch_dist_10 (Table 6.8)	7.0710	6.6445	18.2302	9.7337	10.4343

Table 6.3: Distance values between chroma vector [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1] and some chord interpretations.

For reference, Table 6.3 shows the distances between chroma vector [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1] and several chord interpretations using the values of Tables 6.1, 6.5, 6.6, 6.7, and 6.8. You can see that as the model becomes more complex, the number of cases it can distinguish increases. Note that **basicspace** can distinguish the same cases as **ch_dist_5**.

6.3 From Chroma Vectors to Chod Interpretation Paths

The method explained in Sections 3 and 4 receives chord symbols as the input, but now we modify it to receive chroma vectors. The graph structure becomes like Figure 6.2. The layer width becomes 24 (keys) $\times$ 7 (degrees) = 168 because all interpretations should be considered at every layer. Then every layer is duplicated to accept chroma vector inputs. Nodes in duplicated layers are connected by horizontal edges whose weights express the distances in the models introduced in Section 6.2. Figure 6.2 shows a graph $G^{(cd)}$ in which each layer of the original graph G has been duplicated and the corresponding nodes are connected to each other by (horizontal) directed edges.

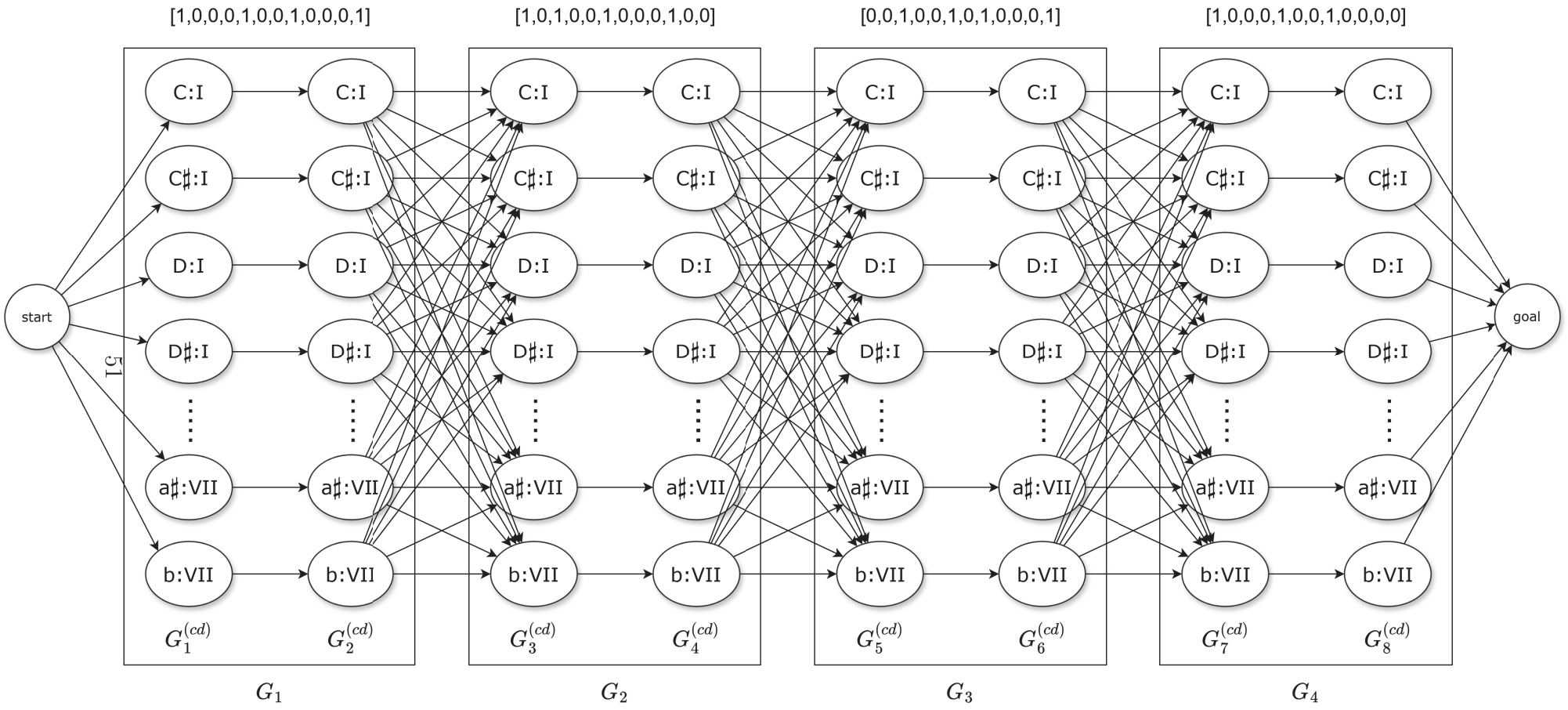

Figure 6.2: Extended interpretation graph.

The learnable parameters are trained to maximize the path probability of the ground-truth paths. However, the formula of path probability (Equation (4.1)) is revised as follows because of the modifications in the interpretation graph.

$$\begin{aligned}
&\mathrm{P}(X_{1:t} = x_{1:t} | c_{1:t}, G_{1:t}) \\
&\triangleq \begin{cases}
1/|G_1| & \text{if } t = 1 \\
(\prod_{u=1}^{t-1} \frac{\exp(-(\boldsymbol{\delta}^{(cd)}(c_u, x_u) + \boldsymbol{\delta}(x_u, x_{u+1}))}{Z_{c,G,u}})) & \\
\times \boldsymbol{\delta}^{(cd)}(c_t, x_t) / Z^{(cd)}_{c,G,t} & \text{otherwise}
\end{cases}
\end{aligned} \tag{6.1}$$

where

$$\begin{aligned}
Z_{c,G,u} &\triangleq \sum_{l \in G_u} \sum_{m \in G_{u+1}} \mathrm{P}(X_u = l | G_{1:u}) \\
&\qquad \times \exp(-(\boldsymbol{\delta}^{(cd)}(c_u, l) + \boldsymbol{\delta}(l, m))), \\
Z^{(cd)}_{c,G,u} &\triangleq \sum_{l \in G_u} \mathrm{P}(X_u = l | G_{1:u}) \exp(-\boldsymbol{\delta}^{(cd)}(c_u, l)),
\end{aligned}$$

x_t is a chord interpretation at t, c_t is a chroma vector at t, $\boldsymbol{\delta}^{(cd)}$ is a chroma distance model. This formula is designed to convert a path length (i.e., $\sum_{u=1}^{t-1}(\boldsymbol{\delta}^{(cd)}(c_u, x_u) + \boldsymbol{\delta}(x_u, x_{u+1}))$) to a path probability so that the shorter (shortest) path has higher (highest) probability just like Equation (4.1). Note that this "length" is the total length in the graph $G^{(cd)}$, but in the formula it is represented with G and chroma vector sequence c for the sake of convenience.

The new graph (Figure 6.2) is a lot more complex than the original graph (Figure 2.5). There are $168 \times 168 = 28,224$ (not $6 \times 6 = 36$) edges between every $G^{(cd)}_{2n}$ and $G^{(cd)}_{2n+1}$ layers, and 168 edges between every $G^{(cd)}_{2n-1}$ and $G^{(cd)}_{2n}$ layers. However, this calculation is not as complicated as it looks. The 28,224 edges that appear over and over again are all from the same, so they can be calculated once and used repetitively. Moreover, the 168 horizontal edges corresponding to a chroma vector can be calculated by matrix product.

6.4 Comparative Analysis

6.4.1 Experimental Setup

We use the dataset annotated in `rntxt` format [38], published at [11]. There are 384 pieces (1,905 phrases, 68,463 chords)[1] and we regard every phrase as a unit

[1]This is different from what is shown in Section 5.1, but this is due to the difference in when the data was obtained.

and we divide it not to exceed 50 chords. Then we use 80% for training, 10% for validation, and the remaining 10% for the test.

We extracted key, degree, and applied chord information from `rntxt`, then omit all repetitions of the same chord interpretations. About applied chords, a tonic chord is added at the end of every local key section to express pivot chord modulation (as the *epsilon* setting in Section 5.1). Chroma vectors are obtained from `rntxt` using music21 library [7].

We set all the initial parameter values to be zero and train them by mini-batch stochastic gradient descent with batch size=100 and learning rate=0.001. We continue training at least 10 epochs and until no accuracy update in the validation set for an epoch[2] then pick the parameter which gives the highest validation accuracy.

In this experiment, we used DE 8.1 from [43] as the (pre-trained) distance model between chord interpretations, which is one of the best performing models with 685#EP achieving 86.25% accuracy[3].

6.4.2 Results and Discussions

Table 6.4 shows the performance of the chroma distance models defined in Section 6.2. The performance is evaluated by how frequently the found path goes through the ground-truth nodes (i.e., chord interpretations) in the revised interpretation graph. **acc** represents the accuracy the method could estimate ground-truth chord interpretation for each chroma vector, while **key** represents the accuracy the method could estimate at the ground-truth key for each chroma vector. For example, if the correct answer is C:I for input [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1], but the prediction result is C:iii, it is treated as an incorrect answer in **acc**, but is counted as a correct answer in **key**.

Even if we used a fairly strong model for the distance model between chord interpretations, it is almost impossible to narrow down the candidates without a hint from chroma vector (i.e., ch_dist_0). Having said that, 3.58% is much better than $1/168 \approx 0.60\%$. Compared to **ch_dist_0**, **ch_dist_2** performed very well with only distinguishing the membership of chords. Also, separating diatonic PCs (**ch_dist_3**), relative positions in triad (**ch_dist_5**), and major or minor key (**ch_dist_10**) all contributed to improve accuracy to some extent. Moreover, the result shows that **basic-space** worked quite well. It went below the most complex model (**ch_dist_10**) but outperformed the other models. This result we think indicates the importance of distinguishing that five categories. The most complex model can be thought as a combination of two revised **basicspace**s for major key

[2]We loosened the stopping condition because the original condition in [43] was too costly to conduct an exhaustive evaluation.

[3]This model is equivalent to **m_sym2** $\times$ **t_min_steps** $\times$ **d_src** $\times$ **d_dest** when using the representation method defined in Chapter 3.

	acc	key acc
basicspace	50.21%	60.14%
ch_dist_0	3.58%	11.22%
ch_dist_2	49.30%	57.07%
ch_dist_3	49.00%	58.67%
ch_dist_5	53.52%	62.86%
ch_dist_10	55.53%	65.67%

Table 6.4: Performance of chroma distance models. **acc** represents the accuracy the method could estimate ground truth chord interpretation (degree and key), while **key** represents the accuracy when only key is concerned.

and minor key, and it achieved the best performance. Thus, it seems to make sense to switch the PC weights for each scale.

Next are the specific distance values obtained from each model[4].

chord	other
0	2.4576

Table 6.5: Resulting PC-level distance values of **ch_dist_2**.

chord	scale	other
0	2.0414	2.6578

Table 6.6: Resulting PC-level distance values of **ch_dist_3**.

root	third	fifth	scale	other
0.8525	2.8191	0	3.1986	4.2753

Table 6.7: Resulting PC-level distance values of **ch_dist_5**.

	root	third	fifth	scale	other
minor	1.2041	2.5036	0	3.3633	3.6971
major	0.7070	2.6557	0.0956	3.1862	6.2754

Table 6.8: Resulting PC-level distance values of **ch_dist_10**.

[4]Distance values **basicspace** is shown in Table 6.1.

Looking at the learned PC-level distance values, "other" category has the largest and "scale" category has the second-largest values. This is consistent with **basicspace** (Table 6.1). But within the "chord" category, "root" has the smallest value in **basicspace** while "fifth" has the smallest in the learned values. The fact that "fifth" has smaller values than "root" is against our intuition. In this experiment, we obtained the chroma vector using music21 library. However, in an unpublished experiment where we obtained the chroma vector directly from MIDI data, "root" had the smallest values. Therefore, there may be an issue with the way to obtain the chroma vector. Further investigation is needed regarding this aspect.

Chapter 7

Conclusion

7.1 Achievements

In this study, we first introduced a framework in which we construct various distance functions based on three basic features of harmony, train the parameters, and evaluate the performances. We tested all simple combinations exhaustively and explored effective ways to combine features to construct the distance functions. Our experimental results showed that (1) accuracy over our chosen test dataset reached a maximum with only a relatively small number of effective parameters (#EP) in each setting, and 6 #EP was sufficient to surpass the score of the original TPS, (2) it was essential to include all three basic features, (3) multiplication of functions, in many cases, only slightly improved the performance despite a significant increase in the number of parameters required, (4) the distinction of forward/ backward progression are also not very effective, (5) the number of steps in region is more important than that in tonic. From the viewpoints of the above (3)-(5), we could confirm that TPS indeed was judiciously designed although our functions outperformed TPS with respect to accuracy in the key-estimation task. In addition, experiments were conducted in three settings (i.e., *original*, *local*, and *epsilon*) concerning secondary dominants, showing (6) an additional chord, supplemented as a pivot chord (ϵ-transition), helps to deal with a secondary chord.

We then compared an obtained model to TPS and some other models for further discussion and had the following observations. In key space, the boundaries of close keys appear more clearly in our model than in TPS and Chew's spiral array model. The area of the close keys generally agrees with related keys (Equation (2.2)), but includes I $\leftrightarrow$ iv instead of parallel keys, excluding I $\leftrightarrow$ v even though they are the same perfect fifth interval. In degree space, T $\leftrightarrow$ D has generally the smallest values, followed by T $\leftrightarrow$ SD and SD $\leftrightarrow$ D. Except for VII, the results are largely consistent with the experimental results of [2].

Next, we extended this framework to include the distance between the chord and its interpretations in the calculation. We constructed distance models by dividing the treatment into PCs, inspired by the structure of the basic space of TPS. Our data-driven model outperformed the basic space's score. In particular, we showed that the major/ minor scale distinction, which was not considered in basic space, also contributes well to accuracy. However, it also confirmed that the structure and the assigned values of basic space were quite appropriate.

7.2 Further Directions

There are many potential directions to extend our approach. First, we should include the notion of chord duration; e.g., a chord with dissonant note would better be regarded as modulation if it lingers for multiple bars, while it could be a borrowed chord if it appears only on one beat. Second, it would be worthwhile to incorporate more detailed chord qualities and tensions. Third, we consider extending the interpretation graph in order to include not just bi-grams but 3- or 4-grams so that it can express and analyze long-term harmonic relationships, or cadences, directly. Fourth, we would like to take advantage of the scale (or magnitude) of distances. Since this study focused on the effectiveness of the shortest path, we have not been able to deal with the implications of the scale of distance, but in path probability, this is an indicator that expresses the uncertainty. Furthermore, we would like to consider a method to find the optimal structure among them, assuming a more complex structure such as a tree structure, instead of a simple shortest path in an interpretation graph.

Bibliography

[1] Balzano, G. (1980). The group-theoretic description of 12-fold and microtonal pitch systems. *Computer Music Journal*, 4, pp. 66–84.

[2] Bharucha, J., & Krumhansl, C. L. (1983). The representation of harmonic structures in music: Hierarchies of stability as a functions of context. *Cognition*, 13(1), pp. 63–102.

[3] Catteau, B., Martens, J., & Leman, M. (2007). A probabilistic framework for audio-based tonal key and chord recognition. In *Proc. of the 30th Annual Conference of the Gesellschaft für Klassifikation*, pp. 637–644. Freie Universität Berlin, Germany.

[4] Chen, T.-P. & Su, L. (2021). Attend to chords: Improving harmonic analysis of symbolic music using Transformer-based models. *Transactions of the International Society for Music Information Retrieval (TISMIR)* 4, 1, pp. 1–13.

[5] Chew, E. (2002). The spiral array: An algorithm for determining key boundaries. In C. Anagnostopoulou, M. Ferrand, & A. Smaill (Eds.), *Music and artificial intelligence* pp. 18–31. Berlin, Germany: Springer.

[6] Cohn, R. (1997). Neo-Riemannian operations, parsimonious, trichords, and their Tonnetz representations. *Journal of Music Theory*, 14(1), pp. 1–66.

[7] Cuthbert, M. S., & Ariza, C. (2010). music21: A toolkit for computer-aided musicology and symbolic music data. In *Proc. of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pp. 637–642. Utrecht, Netherlands.

[8] Deutsch, D., & Feroe, J. (1981). The internal representation of pitch sequences in tonal music. *Psychological Review*, 88, pp. 503–522.

[9] Drobisch, M. (1855). Uber musikalische Tonbestimmung und Temperatur. In *Abhandlungen der Koniglich sachsischen Gesellschaft der Wissenschaften zu Leipzig. Vierter Band: Abhandlungen der mathematisch-physischen Classe. Zweiter Band*, pp. 3–121. Leipzig, Germany: S. Hirzel.

[10] Euler, L. (1739). *Tentamen novae theoriae musicae*. St. Petersburg Academy.

[11] Gotham, M., Kleinertz, R., Weiss, C., Müller, M., & Klauk, S. (2021). What if the 'When' implies the 'What'?: human harmonic analysis datasets clarify the relative role of the separate steps in automatic tonal analysis. In *Proc. of the 22nd International Society for Music Information Retrieval Conference (ISMIR)*, pp. 229–236. Online.

[12] Gotham, M., Micchi, G., López, N. N., & Sailor, M. (2023). When in Rome: A Meta-corpus of Functional Harmony. *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 6(1), pp. 150–166.

[13] Hall, D. (1974). Quantitative evaluation of musical scale tunings. *American Journal of Physics*, 48, pp. 543–552.

[14] Heinichen, J. D. (1728). *General-bass in der composition*. Dresden: J. D. Heinichen.

[15] Harte, C., Sandler, M., & Gasser, M. (2006). Detecting harmonic change in musical audio. In *Proc. of the 1st ACM workshop on Audio and music computing multimedia*, pp. 21–26.

[16] Kellner, D. (1737). *Treulicher Unterricht im General-Bass*. Hamburg: C. Herold.

[17] Krumhansl, C. L. (1990). *Cognitive Foundations of Musical Pitch*. Oxford: Oxford University Press.

[18] Longuet-Higgins, H. C. (1962). Letter to a musical friend. *Music Review*, 23, pp. 244–248.

[19] Longuet-Higgins, H. C. (1962). Second letter to a musical friend. *Music Review*, 23, pp. 271–280.

[20] Lerdahl, F. (2001). *Tonal pitch space*. New York, NY: Oxford University Press.

[21] Lerdahl, F., & Jackendoff, R. (1983). *A Generative Theory of Tonal Music*. Cambridge, MA: MIT Press.

[22] Madjiheurem, S., Qu, L., & Walder, C. (2016) Chord2vec: Learning musical chord embeddings. In *Proc. of the constructive machine learning workshop at 30th conference on Neural Information Processing Systems (NIPS)*, Barcelona, Spain.

[23] Matsubara, M., Kodama, T., & Tojo, S. (2016). Revisiting cadential retention in GTTM. In *Proc. of the 8th International Conference on Knowledge and Systems Engineering (KSE)*, pp. 218–223. Hanoi, Vietnam.

[24] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proc. of Advances in Neural Information Processing Systems (NIPS)*, pp. 3111–3119.

[25] Milne, A. J., & Holland, S. (2016). Empirically testing Tonnetz, voice-leading, and spectral models of perceived triadic distance. *Journal of Mathematics and Music*, 10(1), pp. 59–85.

[26] Piston, W. (1987). *"Harmony"*. W.W. Norton. (Original work published 1948)

[27] Randall, R. R., & Khan, B. (2010). Lerdahl's tonal pitch space model and associated metric spaces. *Journal of Mathematics and Music*, 4(3), pp. 121–131.

[28] Riemann, H. (1902). *Grosse kompositionslehre, Vol. 1.* Berlin: W. Spemann.

[29] Rocher, T., Robine, M., Hanna, P., & Oudre, L. (2010). Concurrent estimation of chords and keys from audio. In *Proc. of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pp. 141–146. Utrecht, Netherlands.

[30] Sakamoto, S., Arn, S., Matsubara, M., & Tojo, S. (2016). Harmonic analysis based on tonal pitch space. In *Proc. of the 8th International Conference on Knowledge and Systems Engineering (KSE)*, pp. 230–233. Hanoi, Vietnam.

[31] Shepard, R. N. (1964). Circularity in judgments of relative pitch. *Journal of the Acoustical Society of America*, 36, pp. 2345–2353.

[32] Shepard, R. N. (1982). Structural representations of musical pitch. In *The psychology of music* (1st ed.), pp. 343–390. New York, NY: Academic Press.

[33] Schoenberg, A. (1969). *Structural functions of harmony* (rev. ed.). New York, NY: Norton. (Original work published 1954)

[34] Temperley, D. (1999). What's Key for Key? The Krumhansl-Schmuckler Key-Finding Algorithm Reconsidered. *Music Perception*, 17, 1, pp. 65–100.

[35] Stumpf, C. (2012). *The origins of music*. Oxford, UK: Oxford University Press. (First published in 1911, translated by David Trippett)

[36] Tymoczko, D. (2006). The geometry of musical chords. *Science*, 313, pp. 72–74.

[37] Tymoczko, D. (2012). The generalized Tonnetz. *Journal of Music Theory*, 56(1), pp. 1–52.

[38] Tymoczko, D., Gotham, M., Cuthbert, M. S., & Ariza, C. (2019). The romantext format: a flexible and standard method for representing Roman numeral analyses. In *Proc. of the 20th International Society for Music Information Retrieval Conference (ISMIR)*, pp. 123–129. Delft, Netherlands.

[39] Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13.2, pp. 260–269.

[40] Weber, G. (1821-1824). *Versuch einer geordneten theorie der tonsetzkunst*. Mainz: B. Schotts Söhne.

[41] Yamaguchi, N., & Sugamura, N. (2010). Improve TPS to tackle non key constituent note. In *Proc. of the 72nd National Convention of IPSJ*, (0), pp. 153–154.

[42] Yamamoto, H., Uehara, Y., & Tojo, S. (2020). Jazz harmony analysis with ϵ-transition and cadential shortcut. In *Proc. of the 17th Sound and Music Computing Conference (SMC)*, pp. 316–322. Turin, Italy.

[43] Yamamoto, H., & Tojo, S. (2021). Generalized tonal pitch space with empirical training. In *Proc. of the 18th Sound and Music Computing Conference (SMC)*, pp. 300–307. Online.

[44] Yamamoto, H., & Tojo, S. (2023). Effective Features for Finding Chord Interpretation Paths. In *Proc. of the Knowledge, Information and Creativity Support Systems (KICSS)*, vol. 003, LIIR079, pp. 1–12. Kyoto, Japan.

[45] Yamamoto, H., & Tojo, S. (2023). Beyond the basic-space of Tonal pitch space: Distance in chords and their interpretation. In *Proc. of 8 th International Conference on Technologies for Music Notation and Representation (TENOR)*, pp. 83–87. Boston, Massachusetts.

[46] Yamamoto, H., & Tojo, S. (2025). Effective Features to Define Harmonic Distance Models. *Journal of New Music Research*. (Accepted)

Appendix A

Details on Calculations and Processing

A.1 Properness of the Path Probability

Theorem 2 (properness)**.** *The path probability (Equation (4.1)) is a proper probability mass function.*

Proof. It is trivial that the right-hand side of Equation (4.1) takes non-negative value, thus we show a proof of it being normalized by induction.

I. Base case: $t = 1$

$$\sum_{x_1 \in G_1} \mathrm{P}(X_1 = x_1 \mid G_1)$$
$$= \sum_{x_1 \in \{\mathrm{start}\}} 1/|\{\mathrm{start}\}| = 1.$$

II. Induction case:

Assume the probability for $x_{1:s}$ is normalized:

$$\sum_{x_{1:s} \in G_{1:s}} \mathrm{P}(X_{1:s} = x_{1:s} \mid G_{1:s}) = 1$$

It follows that:

$$\sum_{x_{1:s+1} \in G_{1:s+1}} \mathrm{P}(X_{1:s+1} = x_{1:s+1} \mid G_{1:s+1})$$
$$= \sum_{x_{1:s+1} \in G_{1:s+1}} \prod_{t=1}^{s} \frac{\exp(-\boldsymbol{\delta}(x_t, x_{t+1}))}{Z_{G,t}}$$

$$= \sum_{x_{1:s+1} \in G_{1:s+1}} \left(\prod_{t=1}^{s-1} \frac{\exp(-\boldsymbol{\delta}(x_t, x_{t+1}))}{Z_{G,t}} \right) \frac{\exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}}$$

$$= \sum_{x_{1:s+1} \in G_{1:s+1}} \mathrm{P}(X_{1:s} = x_{1:s} \mid G_{1:s}) \frac{\exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}} \ (*)$$

$$= \sum_{x_{1:s} \in G_{1:s}} \mathrm{P}(X_{1:s} = x_{1:s} \mid G_{1:s}) \sum_{x_{s+1} \in G_{s+1}} \frac{\exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}},$$

Then, by the assumption, we have

$$= \mathbb{E}_{x_{1:s} \sim \mathrm{P}(X_{1:s} = x_{1:s} | G_{1:s})} \left[\sum_{x_{s+1} \in G_{s+1}} \frac{\exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}} \right]$$

$$= \mathbb{E}_{x_s \sim \mathrm{P}(X_s = x_s | G_s)} \left[\sum_{x_{s+1} \in G_{s+1}} \frac{\exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}} \right]$$

$$= \sum_{x_s \in G_s} \mathrm{P}(X_s = x_s \mid G_{1:s}) \left[\sum_{x_{s+1} \in G_{s+1}} \frac{\exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}} \right]$$

$$= \frac{\sum_{x_s \in G_s} \mathrm{P}(X_s = x_s \mid G_{1:s}) \left(\sum_{x_{s+1} \in G_{s+1}} \exp(-\boldsymbol{\delta}(x_s, x_{s+1})) \right)}{Z_{G,s}}$$

$$= \frac{\sum_{x_s \in G_s} \sum_{x_{s+1} \in G_{s+1}} \mathrm{P}(X_s = x_s \mid G_{1:s}) \exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}}$$

$$= \frac{Z_{G,s}}{Z_{G,s}}$$

$$= 1.$$

(*) When $s = 1$, $1 = \mathrm{P}(X_{1:1} = x_{1:1} | G_{1:1})$. □

A.2 Differentiating the Loss Function

We update the parameters based on the gradient on the parameter space of loss function (Equation (4.2)). The partial differentiation of **loss** (Equation (4.2)) with respect to a parameter i can be calculated as below:

$$
\begin{aligned}
&\frac{\partial}{\partial i}\mathbf{loss}(x_{1:|G|}|G_{1:|G|}) \\
&= \frac{\partial}{\partial i} \sum_{x_{1:|G|}\in G_{1:|G|}} -\mathrm{P}^*(X_{1:|G|} = x_{1:|G|}) \ln \mathrm{P}(X_{1:|G|} = x_{1:|G|}|G_{1:|G|}) \\
&= \frac{\partial}{\partial i} - \ln \mathrm{P}(X_{1:|G|} = x^*_{1:|G|}|G_{1:|G|}) \\
&// \text{ paths can be reduced to } x^*_{1:|G|} \text{ by } P^* \\
&= \frac{\partial}{\partial i} - \ln \prod_{t=1}^{|G|-1} \frac{\exp(-\boldsymbol{\delta}(x^*_t, x^*_{t+1}))}{Z_{G,t}} \\
&= \frac{\partial}{\partial i} - \left(\ln \sum_{t=1}^{|G|-1} \left(\ln \exp(-\boldsymbol{\delta}(x^*_t, x^*_{t+1}))\right) - \ln Z_{G,t} \right) \\
&= \sum_{t=1}^{|G|-1} \left(\frac{\partial}{\partial i}\boldsymbol{\delta}(x^*_t, x^*_{t+1}) + \frac{\frac{\partial}{\partial i} Z_{G,t}}{Z_{G,t}} \right) \\
&= \sum_{t=1}^{|G|-1} \left(\frac{\partial}{\partial i}\boldsymbol{\delta}(x^*_t, x^*_{t+1}) + \frac{\frac{\partial}{\partial i} \sum_{l\in G_t} \sum_{m\in G_{t+1}} \mathrm{P}(X_t = l|G_{1:t}) \exp(-\boldsymbol{\delta}(l,m))}{Z_{G,t}} \right) \\
&= \sum_{t=1}^{|G|-1} \left(\frac{\partial}{\partial i}\boldsymbol{\delta}(x^*_t, x^*_{t+1}) \right. \\
&\quad \left. + \frac{\sum_{l\in G_t} \sum_{m\in G_{t+1}} \left(\begin{array}{l} \left(\frac{\partial}{\partial i}\mathrm{P}(X_t = l|G_{1:t})\right) \exp(-\boldsymbol{\delta}(l,m)) \\ +\mathrm{P}(X_t = l|G_{1:t})(-\frac{\partial}{\partial i}\boldsymbol{\delta}(l,m)) \exp(-\boldsymbol{\delta}(l,m)) \end{array} \right)}{Z_{G,t}} \right)
\end{aligned}
$$

Here, $\mathrm{P}(X_t = l|G_{1:t})$ and $Z_{G,t}$ appear in the calculation process of path probability, so we can reuse the results. Also, $\boldsymbol{\delta}(x^*_t, x^*_{t+1})$ and $\frac{\partial}{\partial i}\boldsymbol{\delta}(x^*_t, x^*_{t+1})$ can be calculated easily, because function $\boldsymbol{\delta}$ is composed of simple sums and products as described

in Chaper 3. Lastly, $\frac{\partial}{\partial i}\mathrm{P}(X_t = l|G_{1:t})$ can be calculated as below (when $t > 1$):

$$
\begin{aligned}
&\frac{\partial}{\partial i}\mathrm{P}(X_t = l|G_{1:t}) \\
&= \frac{\partial}{\partial i} \sum_{x_{1:t} \in G_{1:t-1}\{l\}} \mathrm{P}(X_{1:t} = x_{1:t}|G_{1:t}) \\
&// \ t\text{-th layer has only one node } l \\
&= \frac{\partial}{\partial i} \sum_{x_{1:t} \in G_{1:t-1}\{l\}} \prod_{s=1}^{t-1} \frac{\exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}} \\
&= \frac{\partial}{\partial i} \sum_{x_{1:t} \in G_{1:t-1}\{l\}} \left(\prod_{s=1}^{t-2} \frac{\exp(-\boldsymbol{\delta}(x_s, x_{s+1}))}{Z_{G,s}} \right) \frac{\exp(-\boldsymbol{\delta}(x_{t-1}, x_t))}{Z_{G,t-1}} \\
&= \frac{\partial}{\partial i} \sum_{x_{1:t} \in G_{1:t-1}\{l\}} \mathrm{P}(X_{1:t-1} = x_{1:t-1}|G_{1:t-1}) \frac{\exp(-\boldsymbol{\delta}(x_{t-1}, x_t))}{Z_{G,t-1}} \\
&= \frac{\partial}{\partial i} \sum_{x_{1:t-1} \in G_{1:t-1}} \mathrm{P}(X_{1:t-1} = x_{1:t-1}|G_{1:t-1}) \times \sum_{x_t \in \{l\}} \frac{\exp(-\boldsymbol{\delta}(x_{t-1}, x_t))}{Z_{G,t-1}} \\
&= \frac{\partial}{\partial i} \sum_{x_{t-1} \in G_{t-1}} \mathrm{P}(X_{t-1} = x_{t-1}|G_{1:t-1}) \frac{\exp(-\boldsymbol{\delta}(x_{t-1}, l))}{Z_{G,t-1}} \\
&// \text{ there no need to sum for the } t\text{-th layer since there is only one node} \\
&= \sum_{x_{t-1} \in G_{t-1}} \left(\left(\frac{\partial}{\partial i}\mathrm{P}(X_{t-1} = x_{t-1}|G_{1:t-1}) \right) \frac{\exp(-\boldsymbol{\delta}(x_{t-1}, l))}{Z_{G,t-1}} \right. \\
&\quad + \mathrm{P}(X_{t-1} = x_{t-1}|G_{1:t-1}) \frac{-\frac{\partial}{\partial i}\boldsymbol{\delta}(x_{t-1}, l) \exp(-\boldsymbol{\delta}(x_{t-1}, l))}{Z_{G,t-1}} \\
&\quad \left. + \mathrm{P}(X_{t-1} = x_{t-1}|G_{1:t-1}) \exp(-\boldsymbol{\delta}(x_{t-1}, l)) \left(-\frac{\frac{\partial}{\partial i} Z_{G,t-1}}{(Z_{G,t-1})^2} \right) \right)
\end{aligned}
$$

And when $t = 1$, $\frac{\partial}{\partial i}\mathrm{P}(X_1 = x_1|G_1) = 0$. This calculation also has the recursive structure as the path probability and node probability, so it can be done with the time complexity linear to t.

A.3 How to Compute Accuracy

We show the algorithm to compute the path accuracy in Algorithm 1.

Basically, this algorithm is based on the idea of dynamic programming. It first counts the number of the shortest paths that start from each node by collecting them backward from the goal node (lines 3-7). This process starts from setting 1 to the goal node's path count, and then, moving backward, copying the counts according to the backward links so the counts multiply at every fork. In the end, the start node's count will be the number of all shortest paths of the graph. In the next step, the algorithm confirms the probabilities to reach each node (lines 8-15). Starting from the start node, which has a probability of 1.0, the algorithm distributes the probability by backtracking the backward links (i.e., moving forward in the graph). Finally, it calculates the average probability of all nodes in the ground-truth path (lines 16-19).

[1]The form is different from that of Section 4 for illustration purposes, but it is easy to convert from it.

Algorithm 1 Algorithm for computing the path accuracy

Input: a // a list of ground-truth interpretations for layer indices
G // an interpretation graph, which is a list of list where G[i][j] means the j-th node of the i-th layer in G. Layer 0 and T+1 are start and goal layer, respectively[1]
b // a backward link list computed by means of Viterbi algorithm where b[i][j] is a list of node indices of G[i-1] linked from G[i][j]

Output: accuracy

```
c ← zeroList(sizeOf(G)) // a list of the number of paths from each node to the goal node. Initially zero
p ← zeroList(sizeOf(G)) // a list of the probabilities of each node in each layer. Initially zero
c[T + 1][0] ← 1
for i ← T+1 to 1 step -1 do
    for j in G[i] do
        for k in b[i][j] do
            c[i-1][k] ← c[i-1][k] + c[i][j]
p[0][0] ← 1
for i ← 0 to T - 1 do
    for j in G[i] do
        pathCount ← 0
        for k in G[i+1] where b[i+1][k] contains j do
            pathCount ← pathCount + c[i+1][k]
        for k in G[i+1] where b[i+1][k] contains j do
            p[i + 1][k] ← p[i + 1][k] + p[i][j] × c[i + 1][k] / pathCount
accuracy ← 0
for i ← 1 to T do
    accuracy ← accuracy + p[i][a[i]]
accuracy ← accuracy / T
```

Appendix B

Details on the Experiments

B.1 The Numbers of Combinations

Table B.1: The numbers of combinations for the experiments in Section 5. "X" represents one of the six basic features. We express elemental functions that use two basic features, as if there were a multiplication.

pattern	combinations
-	1
X	4
X×X	16
X+X	6
X×X×X	32
X×X+X	40
X+X+X	4
X×X×X×X	63
X×X×X+X	56
X×X+X×X	73
X×X+X+X	36
X+X+X+X	1
X×X×X×X×X	64
X×X×X×X+X	96
X×X×X+X×X	160
X×X×X+X+X	48
X×X+X×X+X	96
X×X+X+X+X	16
X×X×X×X×X×X	60
X×X×X×X×X+X	64
X×X×X×X+X×X	244
X×X×X+X×X×X	64
X×X×X+X×X+X	176
X×X+X×X+X×X	68
X×X+X×X+X+X	48
X×X+X+X+X+X	4
total	1,540

B.2 Key Distances in Tonnetz

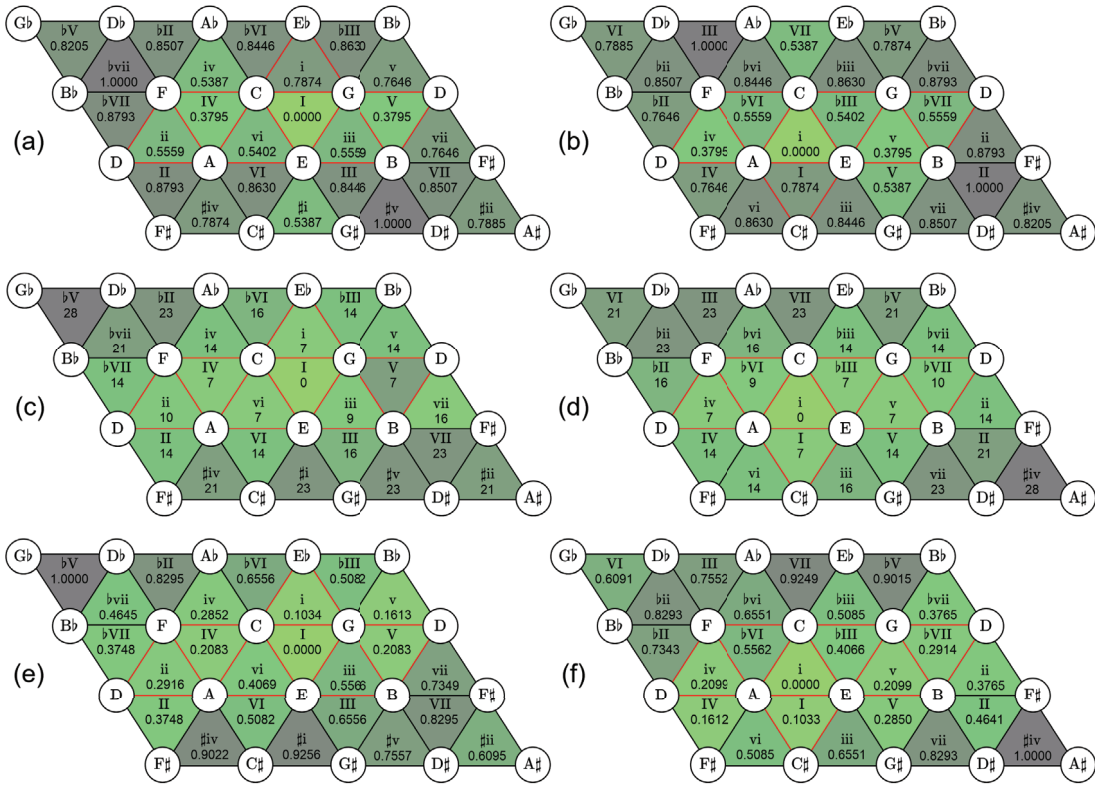

Figure B.1: Distances between each key when degrees are fixed to I. The keys surrounded by red frames are related keys (Equation (2.2)). (a) Distance by our model (Table 5.4 (a)) from major key, (b) from minor key. (c) Distance by **tps** (Equation (2.4)) from major key, (d) from minor key, and (e) distance by Chew's Spiral Array model [5] from major key, (f) from minor key. Keys are arranged in the Tonnetz style.

B.3 Best Models

Here, we enumerate the models (i.e., feature combinations) that performed well and their learned values. Specifically, we enumerate those that have the best accuracy among those with the same or fewer #EP than themselves in the *epsilon* setting (see Figure 5.1).

***always returns 0 (#EP: 0, mean accuracy: 0.1828)**

This one has no learnable parameters.

m_src (#EP: 1, mean accuracy: 0.2563)

Table B.2: Learned parameters of **m_src**. **c1** is **m_src**(x, y)'s index. The values are adjusted to fall within the range of 0 to 1 (the same applies below).

(c1=)0	**1**
1.0000	0.0000

m_desc + d_min_steps (#EP: 4, mean accuracy: 0.2990)

Table B.3: Learned parameters of **m_desc** + **d_min_steps**. (a) shows **m_desc**'s and (b) shows **d_min_steps**'s parameters, where **c1** and **c2** are the index of **m_desc**(x, y) and **d_min_steps**(x, y) respectively.

(a)	**(c1=)0**	**1**		
	1.0000	0.0000		

(b)	**(c2=)0**	**1**	**2**	**3**
	0.6165	0.3016	0.46441	0.0000

m_src + m_dest + d_min_steps (#EP: 5, mean accuracy: 0.3015)

Table B.4: Learned parameters of **m_src + m_dest + d_min_steps**. (a) shows **m_src**'s, (b) shows **m_dest**'s, and (c) shows **d_min_steps**'s parameters, where **c1**, **c2** and **c3** are the index of **m_src**(x, y), **m_dest**(x, y), and **d_min_steps**(x, y) respectively.

(a)	**(c1=)0**	**1**		
	0.6833	0.0000		

(b)	**(c2=)0**	**1**		
	0.6833	0.0000		

(c)	**(c3=)0**	**1**	**2**	**3**
	0.8584	0.7536	1.0000	0.0000

d_dest (#EP: 6, mean accuracy: 0.3936)

Table B.5: Learned parameters of **d_dest**. **c1** is **d_dest**(x, y)'s index.

(c1=)0	**1**	**2**	**3**	**4**	**5**	**6**
0.0000	0.7554	1.0000	0.8299	0.2052	0.9123	0.6986

m_src + t_min_steps (#EP: 7, mean accuracy: 0.5588)

Table B.6: Learned parameters of **m_src + t_min_steps**. (a) shows **m_src**'s and (b) shows **t_min_steps**'s parameters, where **c1** and **c2** are the index of **m_src**(x, y) and **t_min_steps**(x, y) respectively.

(a)	**(c1=)0**	**1**					
	1.0000	0.0000					

(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0000	0.8193	0.6654	0.5653	0.7353	0.4766	0.8328

m_src + m_dest + t_min_steps (#EP: 8, mean accuracy: 0.5601)

Table B.7: Learned parameters of **m_src + m_dest + t_min_steps**. (a) shows **m_src**'s, (b) shows **m_dest**'s, and (c) shows **t_min_steps**'s parameters, where **c1**, **c2** and **c3** are the index of **m_src**(x, y), **m_dest**(x, y), and **t_min_steps**(x, y) respectively.

(a)	**(c1=)0**	**1**					
	0.6236	0.0000					

(b)	**(c2=)0**	**1**					
	0.6374	0.0000					

(c)	**(c3=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0000	1.0000	0.8724	0.6864	0.9586	0.6253	0.9347

m_src × m_dest + t_min_steps (#EP: 9, mean accuracy: 0.5603)

Table B.8: Learned parameters of **m_src × m_dest + t_min_steps**. (a) shows **m_src × m_dest**'s and (b) shows **t_min_steps**'s parameters, where **c1**, **r1** and **c2** are the index of **m_src**(x, y), **m_dest**(x, y), and **t_min_steps**(x, y) respectively.

(a)	**(c1=)0**	**1**					
(r1=)0	0.8701	0.7024					
1	0.7119	0.0000					

(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0000	1.0000	0.9192	0.8314	0.9924	0.7054	0.9143

m_src + t_min_steps + d_min_steps (#EP: 10, mean accuracy: 0.5746)

Table B.9: Learned parameters of **m_src + t_min_steps + d_min_steps**. (a) shows **m_src**'s, (b) shows **t_min_steps**'s, and (c) shows **d_min_steps**'s parameters, where **c1**, **c2** and **c3** are the index of **m_src**(x, y), **t_min_steps**(x, y), and **d_min_steps**(x, y) respectively.

(a)	**(c1=)0**	**1**					
	0.8959	0.0000					
(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0000	0.9439	0.8012	0.6893	0.8190	0.0550	1.0000
(c)	**(c3=)0**	**1**	**2**	**3**			
	0.5719	0.1015	0.2648	0.0000			

m_src + m_dest + t_min_steps + d_min_steps (#EP: 11, mean accuracy: 0.5794)

Table B.10: Learned parameters of **m_src + m_dest + t_min_steps + d_min_steps**. (a) shows **m_src**'s, (b) shows **m_dest**'s, (c) shows **t_min_steps**'s, and (d) shows **d_min_steps**'s parameters, where **c1**, **c2**, **c3**, and **c4** are the index of **m_src**(x, y), **m_dest**(x, y), **t_min_steps**(x, y), and **t_min_steps**(x, y) respectively.

(a)	**(c1=)0**	**1**					
	1.0000	0.0000					
(b)	**(c1=)0**	**1**					
	0.9966	0.0000					
(c)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0000	0.6101	0.5044	0.4049	0.5441	0.3589	0.5966
(d)	**(c3=)0**	**1**	**2**	**3**			
	0.4017	0.0834	0.1856	0.0000			

t_min_steps + d_dest (#EP: 12, mean accuracy: 0.7193)

Table B.11: Learned parameters of **t_min_steps + d_dest**. (a) shows **t_min_steps**'s and (b) shows **d_dest**'s parameters, where **c1** and **c2** are the index of **t_min_steps**(x, y) and **d_dest**(x, y) respectively.

(a)	**(c1=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0000	0.8470	1.0000	0.9355	0.9126	0.7754	0.8004
(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0302	0.4195	0.5811	0.5141	0.0000	0.4798	0.4006

m_sym2 + t_min_steps + d_dest (#EP: 13, mean accuracy: 0.7979)

Table B.12: Learned parameters of **m_sym2 + t_min_steps + d_dest**. (a) shows **m_sym2**'s, (b) shows **t_min_steps**'s, and (c) shows **d_dest**'s parameters, where **c1**, **c2** and **c3** are the index of **m_sym2**(x, y), **t_min_steps**(x, y), and **d_dest**(x, y) respectively.

(a)	**(c1=)0**	**1**					
	0.0000	0.5327					
(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0000	0.8977	1.0000	0.8917	0.9475	0.7641	0.8229
(c)	**(c3=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0785	0.6403	0.8439	0.6326	0.0000	0.7681	0.4788

m_sym2 × r_min_steps + d_dest (#EP: 19, mean accuracy: 0.8148)

Table B.13: Learned parameters of **m_sym2 × r_min_steps + d_dest**. (a) shows **m_sym2 × r_min_steps**'s and (b) shows **d_dest**'s parameters, where **c1**, **r1** and **c2** are the index of **m_sym2**(x, y), **r_min_steps**(x, y), and **d_dest**(x, y) respectively.

(a)	**(c1=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r1=)0	0.0000	0.7571	1.0000	0.8626	0.7883	0.6042	0.7478
1	0.8017	0.7760	0.9587	0.8569	0.6485	0.8178	0.7460
(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0871	0.3933	0.6146	0.4161	0.0000	0.4886	0.2629

m_sym2 × t_min_steps + d_src + d_dest (#EP: 25, mean accuracy: 0.8201)

Table B.14: Learned parameters of **m_sym2 × t_min_steps + d_src + d_dest**. (a) shows **m_sym2 × t_min_steps**'s, (b) shows **d_src**'s, and (c) shows **d_dest**'s parameters, where **c1**, **r1**, **c2** and **c3** are the index of **m_sym2**(x, y), **t_min_steps**(x, y), **d_src**(x, y), and **d_dest**(x, y) respectively.

(a)	**(c1=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r1=)0	0.0000	0.8100	1.0000	0.9351	0.8991	0.5100	0.7876
1	0.8815	0.8958	0.6517	0.6324	0.7870	0.6984	0.8759

(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0000	0.2099	0.3355	0.1984	0.0986	0.2319	0.1688

(c)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0942	0.1935	0.3308	0.2711	0.0000	0.2506	0.1178

m_sym2 + r_min_steps + d_sym2 (#EP: 28, mean accuracy: 0.8297)

Table B.15: Learned parameters of **m_sym2 + r_min_steps + d_sym2**. (a) shows **m_sym2**'s, (b) shows **r_min_steps**'s, and (c) shows **d_sym2**'s parameters, where **c1**, **c2**, **c3**, and **r3** are **m_sym2**(x, y)'s index, **r_min_steps**(x, y)'s index, y^{degree}, and x^{degree} respectively. Cells that simply duplicate the values of other cells are enclosed in parentheses (the same applies below).

(a)	**(c1=)0**	**1**					
	0.0000	0.4456					

(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
	0.0000	1.0000	0.9817	0.9425	0.6059	0.4947	0.8204

(c)	**(c3=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r3=)0	0.6863	0.2171	0.4922	0.2278	0.0000	0.3460	0.1137
1	(0.2171)	(0.6863)	0.5617	0.4861	0.2261	0.5005	0.4190
2	(0.4922)	(0.5617)	(0.6863)	0.4402	0.5075	0.5530	0.5702
3	(0.2278)	(0.4861)	(0.4402)	(0.6863)	0.2483	0.4977	0.4202
4	(0.0000)	(0.2261)	(0.5075)	(0.2483)	(0.6863)	0.2674	0.2907
5	(0.3460)	(0.5005)	(0.5530)	(0.4977)	(0.2674)	(0.6863)	0.4223
6	(0.1137)	(0.4190)	(0.5702)	(0.4202)	(0.2907)	(0.4223)	(0.6863)

m_sym2 × t_min_steps + d_sym2 (#EP: 34, mean accuracy: 0.8430)

Table B.16: Learned parameters of **m_sym2 × t_min_steps + d_sym2**. (a) shows **m_sym2 × t_min_steps**'s and (b) shows **m_sym2**'s parameters, where **c1**, **r1**, **c2**, and **r2** are **t_min_steps**(x, y)'s index, **m_sym2**(x, y)'s index, y^{degree}, and x^{degree} respectively.

(a)	**(c1=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r1=)0	0.0000	0.7888	1.0000	0.8813	0.8124	0.5989	0.7692
1	0.8245	0.8879	0.7871	0.7649	0.8951	0.7897	0.8197

(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r2=)0	0.6604	0.2283	0.4366	0.2299	0.0000	0.2873	0.1150
1	(0.2283)	(0.6604)	0.4344	0.3962	0.2737	0.3976	0.3486
2	(0.4366)	(0.4344)	(0.6604)	0.4293	0.4282	0.4455	0.4457
3	(0.2299)	(0.3962)	(0.4293)	(0.6604)	0.2837	0.4163	0.3908
4	(0.0000)	(0.2737)	(0.4282)	(0.2837)	(0.6604)	0.3280	0.2901
5	(0.2873)	(0.3976)	(0.4455)	(0.4163)	(0.3280)	(0.6604)	0.3943
6	(0.1150)	(0.3486)	(0.4457)	(0.3908)	(0.2901)	(0.3943)	(0.6604)

m_sym2 × t_min_steps + d_sym1 (#EP: 40, mean accuracy: 0.8528)

Table B.17: Learned parameters of **m_sym2 × t_min_steps + d_sym1**. (a) shows **m_sym2 × t_min_steps**'s and (b) shows **m_sym1**'s parameters, where **c1**, **r1**, **c2**, and **r2** are **t_min_steps**(x, y)'s index, **m_sym2**(x, y)'s index, y^{degree}, and x^{degree} respectively.

(a)	**(c1=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r1=)0	0.0000	0.8677	1.0000	0.9545	0.9284	0.5056	0.8389
1	0.8063	0.9118	0.6462	0.6027	0.7578	0.7097	0.9682

(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r2=)0	0.6833	0.1508	0.3472	0.2095	0.0000	0.2135	0.0823
1	(0.1508)	0.8903	0.4451	0.3506	0.1900	0.4174	0.2923
2	(0.3472)	(0.4451)	0.5841	0.3522	0.3686	0.4849	0.4695
3	(0.2095)	(0.3506)	(0.3522)	0.9285	0.2161	0.3807	0.3246
4	(0.0000)	(0.1900)	(0.3686)	(0.2161)	0.2905	0.2146	0.2426
5	(0.2135)	(0.4174)	(0.4849)	(0.3807)	(0.2146)	0.7120	0.3133
6	(0.0823)	(0.2923)	(0.4695)	(0.3246)	(0.2426)	(0.3133)	0.3746

m_sym2 × r_min_steps + d_src × d_dest (#EP: 61, mean accuracy: 0.8540)

Table B.18: Learned parameters of **m_sym2 × r_min_steps + d_src × d_dest**. (a) shows **m_sym2 × r_min_steps**'s and (b) shows **d_src × d_dest**'s parameters, where **c1**, **r1**, **c2**, and **r2** are **r_min_steps**(x, y)'s index, **m_sym2**(x, y)'s index, y^{degree}, and x^{degree} respectively.

(a)	**(c1=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r1=)0	0.0000	0.8558	0.9185	0.9048	0.8750	0.4398	0.8289
1	0.5711	1.0000	0.8271	0.8234	0.5817	0.5950	0.8227

(b)	**(c2=)0**	**1**	**2**	**3**	**4**	**5**	**6**
(r2=)0	0.6753	0.0650	0.2643	0.1197	0.0000	0.1217	0.1384
1	0.3474	0.9110	0.4694	0.4248	0.1264	0.4758	0.2952
2	0.4817	0.4019	0.5883	0.3820	0.2961	0.4793	0.4700
3	0.3276	0.3338	0.3437	0.9275	0.1453	0.4675	0.3249
4	0.0816	0.4419	0.4761	0.3738	0.3187	0.3458	0.2484
5	0.4766	0.3928	0.4542	0.3702	0.1659	0.7093	0.3136
6	0.0996	0.4407	0.4511	0.4208	0.2848	0.3997	0.3743

m_sym2 × t_oneway_steps + d_src × d_dest (#EP: 71, mean accuracy: 0.8548)

Table B.19: Learned parameters of **m_sym2 × t_oneway_steps + d_src × d_dest**. (a) shows **m_sym2 × t_oneway_steps**'s and (b) shows **d_src × d_dest**'s parameters, where **c1**, **r1**, **c2**, and **r2** are **t_oneway_steps**(x, y)'s index, **m_sym2**(x, y)'s index, y^{degree}, and x^{degree} respectively.

(a)	(c1=)0	1	2	3	4	5	6	7	8	9	10	11
(r1=)0	0.0000	0.8130	0.9409	0.9411	0.9253	0.4498	0.8331	0.4453	0.8104	0.8973	0.9751	0.8064
1	0.7667	0.8607	0.6013	0.6083	0.7683	0.6630	0.9568	0.6638	0.7308	0.5916	0.6186	0.8524

(b)	(c2=)0	1	2	3	4	5	6					
(r2=)0	0.7244	0.0844	0.2917	0.1766	0.0000	0.1149	0.1760					
1	0.3896	0.9762	0.5195	0.4766	0.1629	0.5256	0.3207					
2	0.5353	0.4566	0.6536	0.4281	0.3358	0.5312	0.5250					
3	0.3736	0.3603	0.3966	1.0000	0.1802	0.5089	0.3620					
4	0.1115	0.4927	0.5356	0.4179	0.3596	0.3873	0.2868					
5	0.5139	0.4457	0.5100	0.4025	0.1949	0.7734	0.3503					
6	0.1078	0.4908	0.5054	0.4782	0.3218	0.4492	0.4226					

m_sym2 × t_min_steps × d_sym2 (#EP: 307, mean accuracy: 0.8557)

Omitted because it is too large.

m_sym2 × t_min_steps × d_sym1 (#EP: 391, mean accuracy: 0.8578)

Omitted because it is too large.

m_sym2 × t_oneway_steps × d_sym2 (#EP: 527, mean accuracy: 0.8740)

Omitted because it is too large.

m_sym1 × t_oneway_steps × d_sym1 (#EP: 1,007, mean accuracy: 0.8807)

Omitted because it is too large.

Appendix C

Publications

Journal paper

[1] Yamamoto, H., & Tojo, S. (2025). Effective Features to Define Harmonic Distance Models. *Journal of New Music Research*. (Accepted)

International conferences

[2] Yamamoto, H., Uehara, Y., & Tojo, S. (2020). Jazz harmony analysis with ϵ-transition and cadential shortcut. In *Proc. of the 17th Sound and Music Computing Conference (SMC)*, pp. 316–322.

[3] Yamamoto, H., & Tojo, S. (2021). Generalized tonal pitch space with empirical training. In *Proc. of the 18th Sound and Music Computing Conference (SMC)*, pp. 300–307.

[4] Yamamoto, H., & Tojo, S. (2023). Effective Features for Finding Chord Interpretation Paths. In *Proc. of the Knowledge, Information and Creativity Support Systems (KICSS)*, vol. 003, LIIR079, pp. 1–12. (Outstanding Student Paper Award)

[5] Yamamoto, H., & Tojo, S. (2023). Beyond the basic-space of Tonal pitch space: Distance in chords and their interpretation. In *Proc. of 8 th International Conference on Technologies for Music Notation and Representation (TENOR)*, pp. 83–87.