

國立臺灣大學電機資訊學院電機工程學系

碩士論文 (初稿)

Department of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis (DRAFT)



利用結構性支撐向量機的具音樂表現能力之半自動電腦演奏  
系統

A Semi-automatic Computer Expressive Music Performance  
System Using Structural Support Vector Machine

呂 行

Shing Hermes Lyu

指導教授：鄭士康博士

Advisor: Shyh-Kang Jeng, Ph.D.

中華民國 103 年 6 月

June, 2014



國立臺灣大學  
電機工程學系

碩士論文  
(初稿)

利用結構性支撐向量機的具音樂表現能力之半  
自動電腦演奏系統

呂  
行  
撰



# 國立臺灣大學（碩）博士學位論文 口試委員會審定書

論文中文題目  
論文英文題目

本論文係呂行君（R01921032）在國立臺灣大學電機工程學研究所完成之碩士學位論文，於民國 103 年○○月○○日承下列考試委員審查通過及口試及格，特此證明

口試委員：

\_\_\_\_\_（簽名）

（指導教授）

_____	_____
_____	_____
_____	_____
_____	_____

系主任、所長 \_\_\_\_\_（簽名）

（是否須簽章依各院系所規定）



## 致謝



# 中文摘要

請打開並編輯[abstractCH.tex](#)

關鍵字：壹、貳、參、肆、伍、陸、柒



# Abstract

Open and edit [abstractEN.tex](#)

Key words:A, B, C, D, E, F, G



# Table of Contents

口試委員會審定書	i
致謝	ii
中文摘要	iii
Abstract	iv
Table of Contents	v
List of Figures	viii
List of Tables	ix
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Goal and Contribution . . . . .	2
1.3 Chapter Organization . . . . .	3
<b>2 Previous Works</b>	<b>4</b>
2.1 Varying Goals and Evaluation . . . . .	4
2.2 Researches Classified by Methods . . . . .	6
2.3 Additional Specialties . . . . .	8
<b>3 Proposed Method</b>	<b>10</b>
3.1 Overview . . . . .	10

3.2	A Brief Introduction to SVM-HMM . . . . .	11
3.3	Learning Performance Knowledge . . . . .	17
3.3.1	Training Sample Loader . . . . .	18
3.3.2	Features Extraction . . . . .	18
3.3.3	SVM-HMM Learning . . . . .	19
3.4	Expressive Performance . . . . .	21
3.4.1	SVM-HMM Generation . . . . .	22
3.4.2	MIDI Generation . . . . .	22
3.4.3	Audio Synthesis . . . . .	23
3.5	Features . . . . .	23
3.5.1	Score Features . . . . .	23
3.5.2	Performance Features . . . . .	26
3.5.3	Normalizing Onset Deviation . . . . .	27
<b>4</b>	<b>Corpus Preparation</b>	<b>32</b>
4.1	Existing Corpora . . . . .	32
4.2	Corpus Specification . . . . .	33
4.3	Implementation . . . . .	36
4.3.1	Score Preparation . . . . .	36
4.3.2	MIDI Recording . . . . .	37
4.3.3	MIDI Cleaning and Phrase Splitting . . . . .	38
4.4	Results . . . . .	38
<b>5</b>	<b>Experiments, Results and Discussion</b>	<b>41</b>
5.1	Parameter Selection . . . . .	41
5.1.1	Experiment Design . . . . .	41
5.1.2	Results and Discussion . . . . .	42
5.2	Human-like Performance . . . . .	42
5.2.1	Experiment Design . . . . .	42
5.2.2	Results and Discussion . . . . .	42





5.3	Performer Style Reproduction . . . . .	42
5.3.1	Experiment Design . . . . .	42
5.3.2	Results and Discussion . . . . .	42
5.4	Comparing with State-of-the-Art Works . . . . .	43
5.4.1	Experiment Design . . . . .	43
5.4.2	Results and Discussion . . . . .	43
<b>6</b>	<b>Conclusions</b>	<b>44</b>
	<b>Bibliography</b>	<b>44</b>
<b>A</b>	<b>Software Tools Used in This Research</b>	<b>52</b>
<b>B</b>	<b>Using the Expressive Performance Corpus</b>	<b>53</b>





# List of Figures

1.1	From Composer to Performance . . . . .	2
3.1	High Level System Architecture . . . . .	11
3.2	Hidden Markov Model . . . . .	16
3.3	Learning Phase Flow Chart . . . . .	17
3.4	Example input file . . . . .	20
3.5	Performing Phase Flow Chart . . . . .	21
3.6	Interval from/to neighbor notes . . . . .	24
3.7	Relative Duration with the previous/next note . . . . .	25
3.8	Metric position . . . . .	26
3.9	Example Score Features . . . . .	26
3.10	Example Performance Features . . . . .	28
3.11	Problem with Onset Deviation . . . . .	28
3.12	Normalization Schemes . . . . .	29
3.13	The Effect of Automatic Normalization on Onset Deviation feature . . . . .	31
4.1	Example MusicXML score . . . . .	35
4.2	Phrase Length Distribution . . . . .	39
4.3	Example Recording Compared to Score (Pianoroll) . . . . .	40



# List of Tables

2.1	List of Reviewed Systems . . . . .	6
4.1	Clementi's Sonatinas Op. 36 . . . . .	34
4.2	Phrases and Notes Count for Clementi's Sonatina Op. 36 . . . . .	39
4.3	Performer Music Experience . . . . .	40
4.4	Total Recorded Phrases and Notes Count . . . . .	40



# Chapter 1

## Introduction

TODO:need rewrite

### 1.1 Motivation

Computer generated music, such as synthesized MIDI, are often considered robotic and unexpressive. But we have already witnessed the fluid and lively sound generated by state-of-the-art text-to-speech systems. This inspired us to develop a system that can read a music score and play it in an expressive, humanly way. Such system can be used for audiolizing score notation editing software, creating interactive media content, and generating royalty-free music.

Established pianists always has his/her own distinctive style. Such style distinguished himself/herself from all the other pianists. If the expressive performance system can learn the style of a performer, it might be able to provide musicological insight of performance styles. Furthermore, we can even make a master who is no longer with us play music he/she never played in his/her lifetime.

TODO:Discuss Mazolla's theory of expressive performance

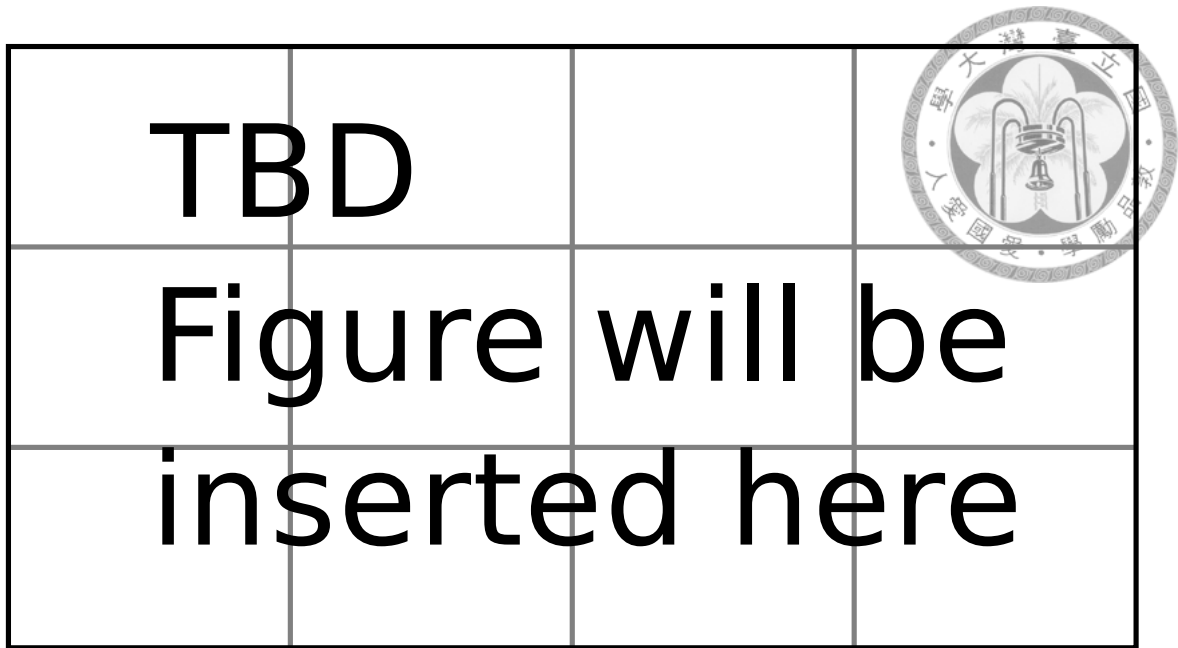


Figure 1.1: From Composer to Performance

## 1.2 Goal and Contribution

TODO:brief guide to previous works The ultimate goal of a expressive performance system is to be able to play any music in a expressive and human-like manner, or even have creative expressions. But due to the technical and time constrain, we need to define a more practical goal for our research. We wish to build a computer expressive performance system based on offline supervised learning. The system will be able to learn any play monophonic melodic phrases. By learning from human recordings, the system will be able to generate expressive performance from previous unseen music notation (also in phrases). The performance style will be controlled by the learning material, if a lot of recordings from different performers are fed in, the system should be able to perform a general human-like expression. If only recordings from a single performer is given, it should learn the particular style of the performer.

The major contribution of this thesis is applying structural support vector machine on expressive performance problem. Also we tried to provide a public corpus for expressive performance. TODO:define scope

## 1.3 Chapter Organization

TODO: Overview of the chapter sctructure of this thesis





## Chapter 2

### Previous Works

Researches on computer expressive music performance lags computer composition by almost a quarter of a century, starting around the end of the 1980s [1]. Early work includes the KTH system [1]. Reviewing computer expressive performance systems is a hard task, because there are exists large difference in the goals they wants to achieve, which makes evaluation and comparison difficult. To make things worse, although there is a contest called RenCon [2] in which researches compete their performances, there is no training and benchmarking corpus available, so if a system has never entered the contest and doesn't make its source code available, there is no way to compare them.

In this chapter, we will follow the categories summarized in [1]. Readers are suggest to refer to [1] because it gives very detailed discussion on all the systems. First we will discuss the various goals of computer expressive performance. Second, systems will be reviewed by their methods used. Finally, we will highlight some systems with special features.

#### 2.1 Varying Goals and Evaluation

The general goal of a computer expressive performance system is to generate expressive music, as opposed to the robotic and nu-human sound of rendered MIDI or other digital score format. But the definition of “expressive” is ambiguous. The following are the most popular goals a computer expressive performance system wants to achieve:

1. Reproduce human performance.
2. Perform music notations in a non-robotic way.
3. Accompany a human performer.
4. Validate musicological models of expressive performance.
5. Directly render computer composed music works.



Systems that are designed to reproduce human performance usually has a certain performer in mind, like the Zenph re-performance CD [3] which will reproduce the performance style of Rachimaninov, it can even use Rachimaninov's style to perform pieces the musician never played in his lifetime.

Some systems try to perform music notations in a non-robotic way, without a certain style in mind. These systems has been employed in music typesetting softwares, like Sibelius [4], to play the typesetted notation.

Accompaniment systems try to render expressive music that act as an accompaniment for human performer. This kind of systems has great value in music education. The challenge is that the system must be able to track the progress of a huamn performance, which is itself another established topic called score following. And the rendering must be done in real-time, which is uncommon in the field of computer expressive music performance.

The next goal is to validate musicological models. Musicologist may develop theories on music performance expressivity, and wants to validate them by experiments. These system focus more on the specific features that the theory tries to explain, so may not cover every aspect of performance.

Finally, some systems combines computer composition with performance. The benefit of this approach is that the performance module can understand the intention of the composition without the need to interpret them from the notation. These systems usually has their own data structure to represent music, which can contain more information than traditional music notation, but the resulting performance system is not backward compatible.



TBD	TBD
TBD	TBD

Table 2.1: List of Reviewed Systems



The goals discussed above imply different level of musical creativity. Human performance reproduction requires mimicry over creativity, on the other hand, system that plays its own composition can have a large range of creativity to explore. The methods employed also limits the ability of creativity, which will be discussed in the next section.

TODO: Discuss works that focus on timber only, e.g. Prof. Su's violin work

## 2.2 Researches Classified by Methods

Dispite the difference between goals of different expressive performance systems. All of them needs some way to create and apply performance knowledge on unexpressive music. The performance can come from predefined rules or being learned.

Using rules to generate expressive music is the earliest approach tried. Director Musices [5] being one of the early works that is still a living project now. Most of the above systems focus on expressive attributes like note onset, note duration and loudness. But Hermode Tuning System [6] focus more on intonation, thus it can generate expressions that requires string instruments techniques. Pop-E [7] is also a rule-based system which can generate polyphonic music, using its synchronization algorithm to synchronize voices. Computational Music Emotion Rule System [8] pust more emphasis on rules that express human emotions. Other systmes like Hierarchical Parabola System [5] [9] [10] [11], Composer Pulse System [12, 13], Bach Fugue System [14], Trumpet Synthesis System [15, 16] and Rubato [17, 18] are also some systems that use rules to generate expressive performance. Rule-based systems are effective and don't require a long training period before use. But some of the performance nuance may be hard to describe in rules, so there is a natural limit on how complex the rule-based system can be. Lack of creativity is also a problem for rule-based approach.

Another approach is to acquire performance knowledge by learning. Many machine

learning methods have been applied to expressive performance problem. One of the easiest form is to use linear regression, systems like Music Interpretation System [19--21] and CaRo [22--24] both use linear regression to learn performance knowledge. But assuming the expressive performance as a linear system is clearly not true. So Music Interpretation System use try to solve it by using AND operations on linear regression results to achieve non-linearity. But still linear regression is too simple for this highly non-linear problem.

Many other learning algorithms have been tested with success: ANN Piano [25] and Emotional flute [26] uses artificial neural network. ESP Piano [27] and Music Plus One [28--30] uses Statistical Graphical Models, although the later one focus more on accompaniment task rather than rendering notation. KCCA Piano System [31] uses kernel regression. And Drumming System [32] tried different mapping models that generates drum patterns.

Evolutionary computation has also been applied, genetic programming is used in Genetic Programming Jazz Sax [33]. Other examples include the Sequential Covering Algorithm Genetic Algorithm [34], Generative Performance Genetic Algorithm [35] and Multi-Agent System with Imitation [36, 37]. Evolutionary computation takes long training time, and the results are unpredictable. But unpredictable also means there are more room for performance creativity, so these system can create unconventional but interesting performances.

TODO: Discuss works that focus on timber only, e.g. Prof. Su's violin work

Another approach is to use case-based reasoning to generate performance. SaxEx [38--40] use fuzzy rules based on emotions to generate Jazz saxophone performance. Kagurame [41, 42] style (Baroque, Romantic, Classic etc.) instead of emotion. Ha-Hi-Hun [43] takes a more ambitions approach, it's goal is to generate a piece X in the style of and expressive performance example of another piece Y. Another series of researches done by Widmer et al. called PLCG [?, 44, 45] uses data-mining to find rules for expressive performance. It's successor Phrase-decomposition/PLCG [46] added hierarchiacal phrase structures support to the original PLCG system. And the latest research in the series called DISTALL [?, ?] added hierarchical rules to the original one.

Most of the of the performance systems discussed above takes digitalized traditional musical notation (MusicXML etc.) or neutral audio as input. They have to figures out the expressive intention of the composer by musical analysis or assigned by the user. But the last category of computer expressive performance we will discuss here has a great advantage over the previous ones, by combining computer composition and performance, the performance part of the system can directly understand the intention of the composition. Ossia [?] and pMIMACS [?] are two examples of this category. This approach provides great possibility for creativity, but they can only play their own composition, which is rather limited.

TODO:Fig.:selected figures from previous works

## 2.3 Additional Specialties

Most expressive performance systems generates piano performance, because it's relatively easy to collect samples for piano. Even if no instrument is specified, the final performance will often be rendered using piano timbre. Some systmes generates music in other instruments, such as saxophone [38--40], trumpet [15, 16], flute [26] and drums [?]. These systems requires additional model for the instruments, as a result, they can produce expressions that requires instrumental skills.

The genre of music that a system plays is also a special feature one might have. For systems that doesn't specify the genre, usually western tonal music will be the genre of choice. Composers like Mozart, Chopin and so on well accepted by the public, their scores and literatures are easily accessable. However, both saxophone-based works choose Jazz music, because saxophone is an iconic instrument in Jazz performance. The Bach Fugue System [14], obviously, focus on fugue works composed by bach.

The ability to perform polyphonic music is also a rare feature of computer expressive performance systems. Performing polyphonic music requires sincronization between voices, while allowing each voice to have their own expression. Pop-E [7] use a sincronization mechniasm to acheive polyphonic performance. Bach Fugue System [14] is created using the polyphonic rules in music theory about fugue, so it's inherently able to

play polyphonic fugue. KCCA Piano System [31] can generate homophonic music -- an upper melody with an accompaniment -- which is common in piano music. Music Plus One [28--30] is a little bit different because it's an accompaniment system, it adapts non-expressive orchestral accompaniment track to user's performance. Other systems usually generate monophonic tracks only.





## Chapter 3

# Proposed Method

### 3.1 Overview

The high-level architecture of the purposed system is shown in figure 3.1. The system has two phases, the upper half of the figure is the learning phase, while the lower half is the performing phase. In the training phase, score and expressive performance recording pairs are feed in, they will serve as training examples for the machine learning algorithm, namely structural support vector machine with hidden markov model output (a.k.a SVM-HMM). The learning algorithm will produce a performance knowledge model, which can be used to generate expressive performance hereafter. In the performing phase, a score will be given to the system for expressive performance. The SVM-HMM generation module will use the performance knowledge learned in the previous phase to produce expressive performance. The SVM-HMM output then go through a MIDI generator and MIDI synthesizer to produce audible performance.

The system is not intend to add fixed expression to all pieces. Rather it is intended to perform music according to the style which the user wants. This kind of user interactivity can be achieved in two ways: first, the user can choose the training corpus. A user can select a subset of training sample to produce unique models. For example, if the corpus contains only one performer's recordings, the learned model will very likely have a unique style of the performer. Second, the phrasing of a song is given by the user. Since phrasing controls the overall structural interpretation of a music piece, and form a breathing feeling

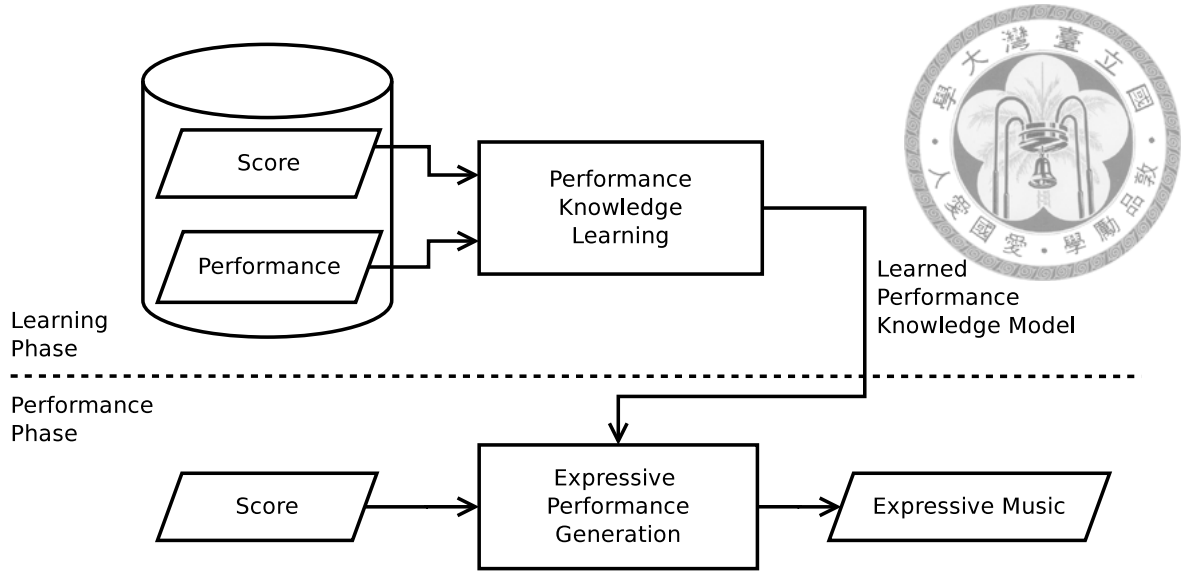


Figure 3.1: High Level System Architecture

in music, user is given direct control over the performance.

There are still some constrains for the system now. The scores must be monophonic and contains only one musical phrase. One has to manually label the phrasing for any scores used. The learning algorithm, namely SVM-HMM, can only perform offline learning, so the learning phase can only work in a non-realtime scenario. The generating phase can work much faster though, it can produce expressive music almost instantaneously after loading the score. All the scores are loaded in batch, the system currently don't accept streaming input.

In the following sections, we will discuss the detail steps in the learning and performing phases, and some implementation detail used in each step. Since SVM-HMM learning requires features to be extracted from samples, we will discuss the features used in the end of this chapter.

## 3.2 A Brief Introduction to SVM-HMM

In this thesis, we use structural support vector machine(structural SVM) as the learning algorithm. Unlike traditional SVM algorithm, which can only produce univariate prediction, structural SVM can produce strctural predictions like tree, sequence and hidden Markov model. Structural SVM with hidden Markov model output (SVM-HMM)

is applied to part-of-speech problem with success. This finding lead us to the idea to use SVM-HMM for the expressive performance problem. The part-of-speech tagging problem shares the same concept with expressive performance problem. In part-of-speech tagging, one tries to identify the role by which the word plays in the sentence, while in expressive performance, one tries to determine how a note should be played, according to it's role in the musical phrase. To illustrate this, consider a note which is the end of the phrase, which is normally forms a cadence, and a note which is only a embellishment. The first note will probably be played louder and sustain longer than the second note. With this similarity in mind, we believe SVM-HMM will be a good candidate for expressive performance.

The prediction problem in SVM can be described as finding a function

$$h : \mathcal{X} \rightarrow \mathcal{Y}$$

with lowest prediction error.  $\mathcal{X}$  is the input features space, and  $\mathcal{Y}$  is the prediction space. In traditional SVM, elements in  $\mathcal{Y}$  are labels (classification) or real values (regression). But structural SVM extends the framework to generate structural output, such as tree, sequence, or hidden markov model, in this case. To extend SVM to support structured output, the problem is modified to find a discriminant function  $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}$ , in which the input/output pairs are mapped to a real number score. To predict an output  $y$  for an input  $x$ , one try to maximize  $f$  over all  $y \in \mathcal{Y}$ .

$$h_{\mathbf{w}}(x) = \arg \max_{y \in \mathcal{Y}} f_{\mathbf{w}}(x, y)$$

Let  $f_{\mathbf{w}}$  be a linear function of the form:

$$f_{\mathbf{w}} = \mathbf{w}^T \Psi(x, y)$$

where  $\mathbf{w}$  is the parameter vector, and  $\Psi(x, y)$  is the kernel function relating input  $x$  to output  $y$ .  $\Psi$  can be defined to accomodate various kind of structures.

In order to apply SVM, we need a way to measure the accuracy of a prediction, in other words, we have to define a loss function  $\Delta : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathcal{R}$ . A loss function has the

following property:

$$\Delta(y, y') \geq 0 \text{ for } y \neq y'$$

$$\Delta(y, y) = 0$$



Also, the loss function is assumed to be bounded. Let's assume the input-output pair  $(x, y)$  is drawn from a joint distribution  $P(x, y)$ , the prediction problem is to minimize the total loss:

$$R_p^\Delta = \int_{\mathcal{X} \times \mathcal{Y}} \Delta(y, f(x)) dP(x, y)$$

Since we can't directly find the distribution  $P$ , we need to replace this total loss with an empirical loss, calculated from the observed training set of  $(x_i, y_i)$  pairs.

$$R_s^\Delta(f) = \frac{1}{n} \sum_{i=1}^n \Delta(y_i, f(x_i))$$

With the definition of the loss function ready, we will demonstrate how to extend SVM to structural output, starting with a linear separable, and then extend it to soft-margin formulation.

A linear separable case can be expressed by a set of linear constraints

$$\forall i \in \{1, \dots, n\}, \forall \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \leq 0$$

However, in the SVM context, we not only want a solution, but we want the solution to have the largest margin possible. So the above linear constraints will become this optimization problem

$$\begin{aligned} & \max_{\gamma, \mathbf{w}: \|\mathbf{w}\|=1} \gamma \\ & \text{s.t. } \forall i \in \{1, \dots, n\}, \forall \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \leq \gamma \end{aligned}$$



, which is equivalent to the convex quadratic programming problem

$$\begin{aligned} \min_{\mathbf{w}, \xi_i \geq 0} & \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ \text{s.t. } & \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq 1 \end{aligned}$$



To address possible non-separable problems, slack variables can be introduced to penalize errors, and result in a soft-margin formalization.

$$\begin{aligned} \min_{\mathbf{w}, \xi_i \geq 0} & \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{n} \sum_{i=1}^n \xi_i \\ \text{s.t. } & \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq 1 - \xi_i \end{aligned}$$

$C$  is the parameter for trade-off between low training error and large margin. The optimal  $C$  varies between different problems, so experiment should be conducted to find the optimal  $C$  for our problem.

Intuitively, a constrain violation with larger loss should be penalize more than the one with smaller loss. So [47] proposed two possible way to take the loss function into account. The first way is to re-scale the slack variable by the inverse of the loss, so a high loss leads to smaller re-scaled slack variable.

$$\begin{aligned} \min_{\mathbf{w}, \xi_i \geq 0} & \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{n} \sum_{i=1}^n \xi_i \\ \text{s.t. } & \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq 1 - \frac{\xi_i}{\Delta(y_i, \hat{y}_i)} \end{aligned}$$

The second way is to re-scale the margin, which yields

$$\begin{aligned} \min_{\mathbf{w}, \xi_i \geq 0} & \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{n} \sum_{i=1}^n \xi_i \\ \text{s.t. } & \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq \Delta(y_i, \hat{y}_i) - \xi_i \end{aligned}$$

But the above quadratic programming problem has a extreme large number ( $O(n|\mathcal{Y}|)$ ) of constrains , which will take considerable time to solve. So [47] proposed a greedy algorithm to reduce the number of constrains. Initially, the solver starts with an empty

working set with no constraints. Then the solver iteratively scans the training set to find the most violated constraints under the current solution. If a constraint violates by more than the desired precision, the constraint is added to the working set. And the solver recalculate the solution under the new working set. The algorithm will terminate once no more constraint can be added under the desired precision.

In a later work by Joachims et al. [48], they created a new formulation and algorithm to further speed up the algorithm. Instead of using one slack variables each training sample, which results in a total of  $n$  slack variables, they use a single slack variable for the  $n$  training samples. The following formula is the 1-slack version of slack-rescaling structural SVM:

$$\begin{aligned} \min_{\mathbf{w}, \xi \geq 0} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C\xi \\ \text{s.t. } \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \quad & \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq \frac{1}{n} \sum_{i=1}^n 1 - \frac{\xi}{\Delta(y_i, \hat{y}_i)} \end{aligned}$$

And margin-rescaling structural SVM:

$$\begin{aligned} \min_{\mathbf{w}, \xi \geq 0} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C\xi \\ \text{s.t. } \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \quad & \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq \frac{1}{n} \sum_{i=1}^n \Delta(y_i, \hat{y}_i) - \xi \end{aligned}$$

Detailed proof on how the new formulation is equivalently general as the old one is given in the paper.

With the framework described above, the only problem left is how to define the general loss function for Hidden Markov Model (HMM)? In [49], the authors proposed two types of features for a equal-length observation/label sequence pair  $(x, y)$ . The first is the interaction of a observation with a label, the other is the interaction between neighboring labels.

Formally, for some observed features  $\Phi_r(x^s)$  of a note  $x$  located in  $s$ th position of the phrase, and assume  $[[y^t = \tau]]$  denotes the  $t$ th note is played at a velocity of  $\tau$ , the

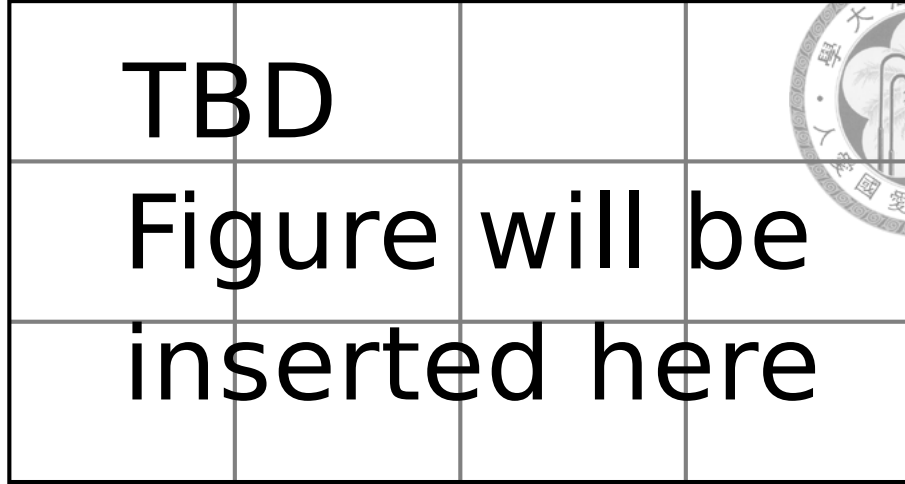


Figure 3.2: Hidden Markov Model

interaction of the two predicate can be written as

$$\phi_{r\sigma}^{st}(\mathbf{x}, \mathbf{y}) = \left[ [y^t = \tau] \right] \Psi_r(x^s), \quad 1 \leq \gamma \leq d, \quad \tau \in \Sigma$$

And for interaction between labels, the feature can be written as

$$\hat{\phi}_{r\sigma}^{st}(\mathbf{x}, \mathbf{y}) = \left[ [y^s = \sigma \wedge y^t = \tau] \right], \quad \sigma, \tau \in \Sigma$$

By selecting a dependency order for the HMM model, we can restrict  $s$ 's and  $t$ 's. For example, for a first-order HMM,  $s = t$  for the first feature, and  $s = t - 1$  for the second feature. The two features on the same time  $t$  is then stacked into a vector  $\Psi(x, y; t)$ . The feature map for the whole sequence is simply the sum of all the feature vectors

$$\Phi(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^T \Phi(\mathbf{x}, \mathbf{y}; t)$$

Finally, the distance between two feature maps depends on the number of common label segments and the inner product between the input features sequence with common labels.

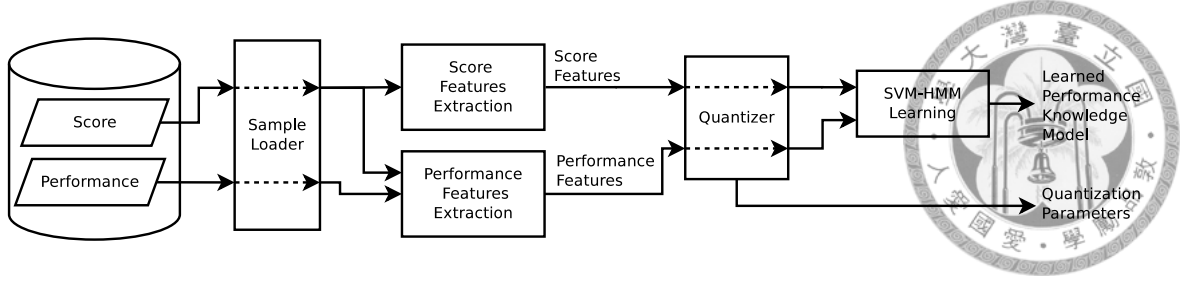


Figure 3.3: Learning Phase Flow Chart

$$\langle \Phi(\mathbf{x}, \mathbf{y}), \Phi(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \rangle = \sum_{s,t} \left[ [y^{s-1} = \hat{y}^{t-1} \wedge y^s = \hat{y}^t] \right] + \sum_{s,t} \left[ [y^s = \hat{y}^t] \right] k(x^s, \hat{x}^t)$$

To speed up the computation of  $F$  for HMM, a Viterbi-like decoding algorithm is used.

### 3.3 Learning Performance Knowledge

The main goal in the learning phase is to extract performance knowledge from training samples. Fig. 3.3 shows the internal structure of the learning phase.

Training samples are matched score and expressive performance pairs (their format and preparation process is discussed in Chapter 4). The training samples are loaded as machine-readable format, and have their features extracted. Two types of features will be extracted from the samples, first, the information from the score only are called score features; second, the information of the expressive performance with respect to the score is called the performance features. In order to generate new expressive performance, we want to learn a prediction model by which we can predict the performance features given the score model. This process can be analogize to a human performer reading the explicit and implicit cues from the score, and perform the music with certain expressive expression. The learning part is done by a supervised learning algorithm. In this thesis, we choose Structural Support Vector Machine with Hidden Markov Model Output (SVM-HMM) as our learning algorithm.

### 3.3.1 Training Sample Loader

A training sample is loaded with the sample loader module, since a training sample is consisted of a score (musicXML format) and an expressive recording (MIDI format), the sample loader finds the two files given the sample name, and load them into an intermediate representation (`music21.Stream` object provided by the `music21` library [?]) from MIT). The `music21` library will convert the musicXML and MIDI format into a python Object hierarchy that is easy to access and manipulate by python code. The loaded score are then handed to the feature extractor module.

One caveat here is that the recording in MIDI may contain very subtle expression. But the `music21` library will quantize the MIDI in the time axis by default, which will destroy the subtle onset and duration expression. And the `music21` library don't handle the “ticks per quarter note” information in the MIDI header [?], so we must explicitly specify that this value, and disable quantization during MIDI loading.

### 3.3.2 Features Extraction

In order to keep the system architecture simple, a feature extractor are designed to be independent to other feature extractors, so features can added or removed without affecting the rest of the system. And further more, this enables parallel processing during feature extraction. To achieve this, each feature must implement a common feature extractor interface, and feature extractors should not communicate with each other.

But sometimes a feature inevitably depends on other features, for example, the “relative duration with the previous note” is calculated based on the “duration” feature. But since we want to avoid complex scheduling with dependency, the “duration” feature is extracted during the extraction of the “relative duration” feature, no matter if the “duration” feature is extracted prior of after the “relative duration”. Therefore, the “duration” feature extracted has computed twice. To avoid redundant computation of the feature extractors, we implemented a caching mechanism. Say, the “duration” feature had been computed once, it's value will be cached during this execution session; when the “relative duration” extractor calls the “duration” extractor again, the value is retrieved from cache without re-

calculation. This method can speed up the execution while keeping the system complexity low.

The extracted features are aggregated and stored into a JavaScript Object Notation (JSON) file. By saving the features in a human-readable intermediate file, the researcher can easily look into the file to debug potential problems.



### 3.3.3 SVM-HMM Learning

After all features are extracted, the next step is to learn performance knowledge from the features. In the early stage of this research, we have tried linear regression with limited success [50]. However, the assumption of linearity is an oversimplification. So we switch to Structural Support Vector Machine with Hidden Markov Model Output (SVM-HMM) [47--49] as our supervised learning algorithm.

The SVM-HMM learning module loads the JSON file from the previous stage, and rearrange the features to fit the required input format of the SVM-HMM learner program. However, the features from the previous stage are real numbers, but SVM-HMM only takes discrete output, so quantization is required here. For each feature, the quantizer calculates the overall mean and standard deviation from all training samples. Then the quantizer divides the range between mean minus three standard deviations to mean plus three standard deviations into 1024 intervals. Feature values below mean minus three standard deviations are quantized to the lowest bin, while values larger than mean plus three standard deviations are quantized to the highest bin. The range of three standard deviation and 1024 intervals are chosen by experience, which can be modified to fit different corpus. The mean, standard deviation and number of intervals information are stored in a file for the performing phase to dequantize the output.

The theoretical background of SVM-HMM is already described in Section 3.2, to implement the algorithm we leverage Thorsten Joachims's implementation called  $SVM^{hmm}$  [51].  $SVM^{hmm}$  is an implementation of structural SVMs for sequence tagging [49] using the training algorithm described in [47] and [48]. The  $SVM^{hmm}$  package contains a model training program called `svm_hmm_learn` and a model prediction program called

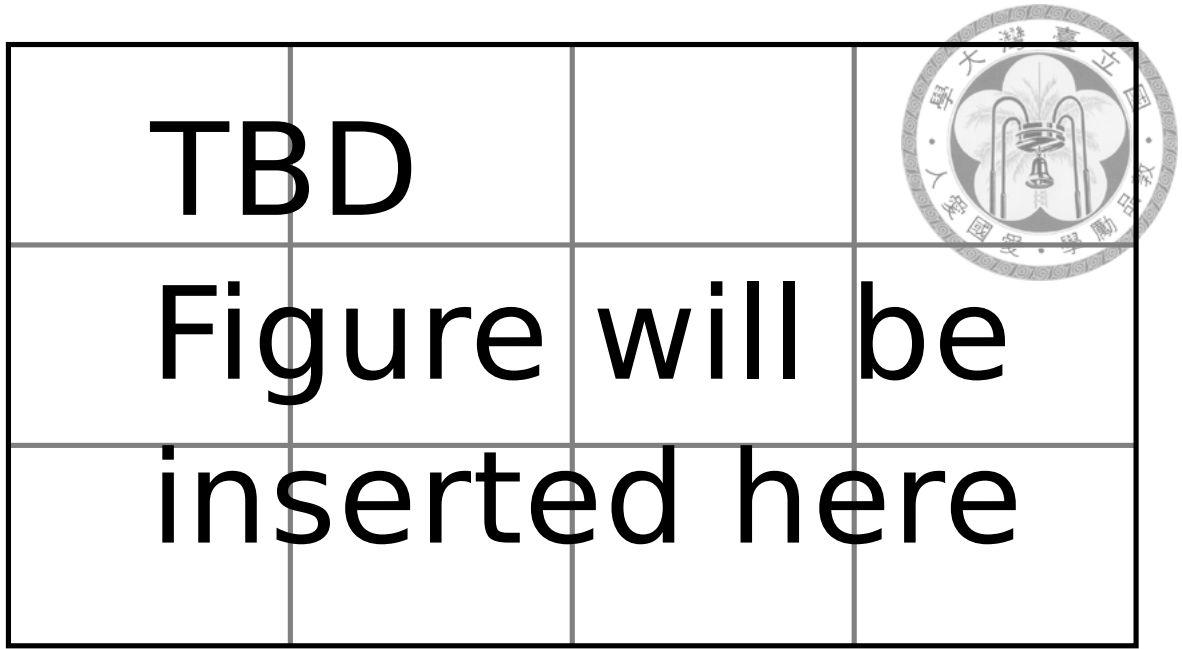


Figure 3.4: Example input file

`svm_hmm_classify`, which will be used in the performing phase. For structural simplicity, we train a separate model for each quantized performance feature, each model uses all the quantized score features to try to predict a single performance model. The `svm_hmm_learn` takes a training file describing those features. Each line represents features for a note, organized in the following format:

```
1      PERF qid:EXNUM FEAT1:FEAT1_VAL FEAT2:FEAT2_VAL ... #comment
```

PERF is a quantized performance feature. The EXNUM after `qid:` identifies the phrases, all notes in a phrase will have the same `qid:EXNUM` identifier. Following the identifier are quantized score features, denote as `feature name : feature value`, separated by space. And anything after the `#` is comments. An example of the training file is shown in Fig. 3.4

TODO: partial model

There are some key parameters need to be specified for the training process. First the  $C$  parameter in SVM, which controls the trade-off between low training error and large margin. Larger  $C$  will result in lower training error, but the margin may be smaller. Second, the  $\epsilon$  parameter controls the required precision for the constrains. The smaller

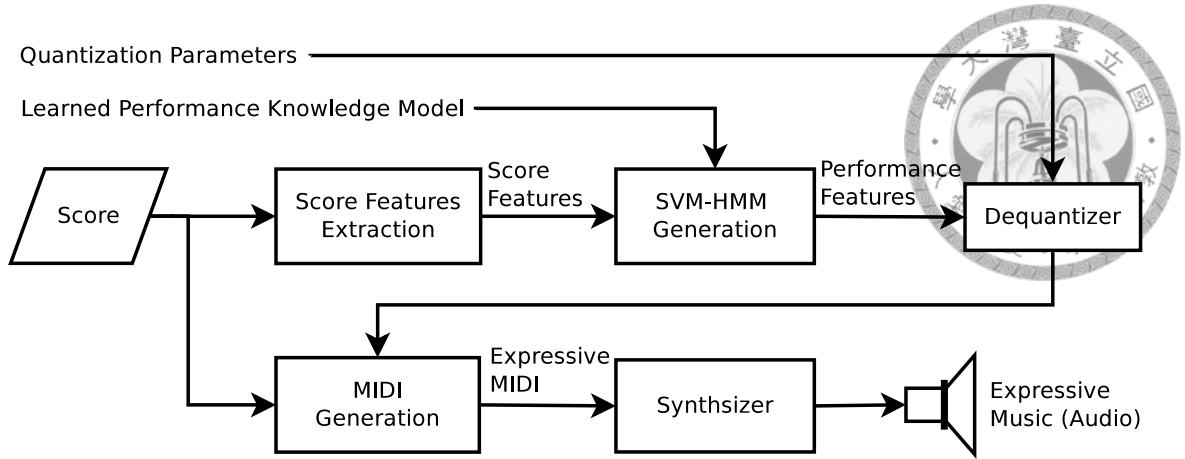


Figure 3.5: Performing Phase Flow Chart

the  $\epsilon$ , the precision will be higher, but may require more computation. Finally, for the HMM part, we need to specify the order of dependencies of transition states and emission states. In our case, transition dependency is set to one, which stands for first-order Markov property, and emission dependency is set to zero. Since we train separate models for each performance feature, each model can have their own set of parameters. The parameter selection process is done by experiment, which will be presented in Chapter 5

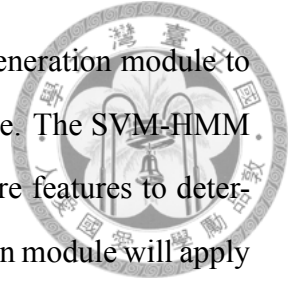
Finally, the training program will output three model files (because we use three performance features). In the model file are binary representation of the SVM-HMM model parameters, such as the support vectors and other informations, which represents the performance knowledge learned. Since it takes considerable time to train a model (depending on the amount of training samples and the power of the computer running the system), the system can only support offline learning. But the learning process only need to be run once. The performance knowledge model can be reused over and over again in the performing phase.

### 3.4 Expressive Performance

With the performance knowledge learned, we can start to perform music expressively. The performing phase share the same sample loader and feature extractor module with the learning phase. But in the performing phase, the input is only a score file (musicXML containing a single phrase), which is loaded and has its score features extracted. The extracted



score features are also stored into a JSON file for the SVM-HMM generation module to load. The performance knowledge model is also an input in this phase. The SVM-HMM generation module will use the learned model and the quantized score features to determine the performance features for the given score. An MIDI generation module will apply those performance features onto the musicXML score to produce a expressive MIDI file. The MIDI file itself is already a expressive performance, in order to listen to the sound, an software synthesizer is used to render the MIDI file into WAV or MP3 format.



### 3.4.1 SVM-HMM Generation

As in the learning phase, the score features are stored in a JSON file. The SVM-HMM generation module first load this JSON file, and preform the same quantization as the learning phase. The quantized score features are then transformed into the same format as the training file, but the PERF fields are all set to zero, meaning that we don't know its value and wish the algorithm to predict it. The `svm_hmm_classify` program will take these inputs with the learned model file and predict the quantized labels of the performance features. If we already know the real performance feature value, say, we use part of the corpus as training set and the rest as testing set, the PERF can be set to the “answer” of the testing set, so the `svm_hmm_classify` program will calculate the accuracy of the prediction.

### 3.4.2 MIDI Generation

de-quantize Since the output of the SVM-HMM learner is the quantized label for the performance feature, dequantization is required to turn those values back to real-valued performance features. The dequantizer will load the quantization parameters from the learning stage to understand the range and intervals used in the quantizer. Each quantization label is dequantized into the mean value of the interval it belongs.

The performance features are than applied onto the input score. For example, if a performance feature represents the note's duration should last for 1.2 times of its nominal value, the duration in the score is multiplied by 1.2. After all the performance features are

applied, the expressive version of the score is stored in MIDI format using the `music21` library.

TODO:Dramatization/post processing



### 3.4.3 Audio Synthesis

In order to actually hear the expressive performance, the MIDI file is then rendered by a MIDI synthesizer. Since the output is standard MIDI file, the user can choose any compatible software or hardware synthesizer. We choose `timidity++` software synthesizer for this job. The output is an WAV(Waveform Audio Format)file, which is then compressed into MP3 (MPEG-2 Audio Layer III) by `lame` audio encoder.

Because sub-note-level expression is not the primary goal of this research, we choose to use standard MIDI synthesizer to render the music. The system can be extended to used more advanced physical model or musical instrument specific audio synthesizer. Sub-note level features, such as special techniques for violins, can be added to the features list and be learned by the SVM-HMM model.

TODO: phrase concatenation

## 3.5 Features

As mentioned in Section 3.3, there are two types of features, score features and performance features. We will present the features used in the system, and discuss the difficulties encountered.

### 3.5.1 Score Features

Score features are musicological cues presented in the score. The purpose of score features are to simulate the high level information a performer may perceive when she reads the score. The basic time unit for the features are notes. Each note will have one of each features presented below. Score features includes:

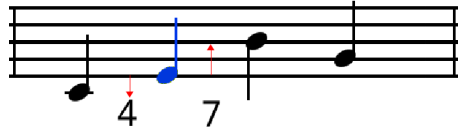


Figure 3.6: Interval from/to neighbor notes

**Relative position in a phrase:** The relative position of a note in the phrase, its value ranges from 0% to 100%. Since there are often salient musical expressions during the opening or closing of a phrase, this feature is used to capture the start or end of the phrase.

**Relative pitch:** The pitch of a note relative to the pitch range of the phrase, denoted by MIDI pitch number (resolution is down to semitone). For a phrase of  $n$  notes with pitch  $P_1, P_2, \dots, P_n$ ,

$$RP = \frac{P_i - \min(P_1, P_2, \dots, P_n)}{\max(P_1, P_2, \dots, P_n) - \min(P_1, P_2, \dots, P_n)}$$

Where  $P_i$  is the pitch of note at position  $t$

**Interval from the previous note:** The interval between the current note and its previous note, in semitone. This represents the direction of the melodic line.

$$IP = P_i - P_{i-1}$$

See Fig. 3.6 for example.

**Interval to the next note:** The interval between the current note and its previous note, in semitone.

$$IN = P_{i+1} - P_i$$

See figure 3.6 for example.

**Note duration:** The duration of a note counted in number of quarter notes.

Although this feature may seem an obvious one, there are still a caveat: Grace notes,

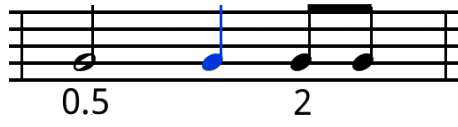


Figure 3.7: Relative Duration with the previous/next note

which represents ornaments, have zero duration in musicXML specification. The reason for this is that grace notes are considered very short ornaments that does not occupy solid beat position. But zero duration will easily cause divide-by-zero exception in the program, and it's hard to handle in math formulation. So we assigned a duration of 0.0625 quarter note to all grace notes, which is equivalent to the length of a sixty-fourth note. Sixty-fourth note is chosen because it's far shorter than all the notes in our corpus.

**Relative Duration with the previous note:** The duration of a note divided by the duration of its previous note. For a phrase of  $n$  notes with duration  $D_1, D_2, \dots, D_n$ ,

$$RDP = \frac{D_i}{D_{i-1}}$$

See figure 3.7 for example. This feature is intended to locate local change in tempo, such as a series of rapid consecutive notes followed by a long note.

**Relative duration with the next note:** The duration of a note divided by duration of its next note.

$$RDN = \frac{D_i}{D_{i+1}}$$

See figure 3.7 for example.

**Metric position:** The position of a note in a measure, measured by the beat unit defined by the time signature. For example, a  $\frac{4}{4}$  time signature will have a beat unit of a quarter note. So if the measure consists of four quarter notes, each of them will have metric position of 1, 2, 3 and 4. Please refer to Fig. 3.8 for example.

Metric position implies beat strength. In most tonal music, there exist a hierarchy



Figure 3.8: Metric position

TBD			
Figure will be			
inserted here			

Figure 3.9: Example Score Features

of beat strength [?]. For example, in a measure of a  $\frac{4}{4}$  piece, the first note is usually the strongest, the third note is the second strongest, and the second and fourth notes are the least strong.

### 3.5.2 Performance Features

Performance features are the expressive expressions of a performance, which is what we want to learn and generate in this research. Performance features are extracted by comparing the expressive performance with the score. Performance features includes:

**Relative onset time deviation:** The onset time of a natural human recording will not be exactly as the ones indicated on the score, this phenomenon is roughly corresponding to the music term “rubato”. Given a fixed tempo (beats per second), the score timing of each note can be calculated as  $\text{tempo} \times (\text{beats from the start of phrase})$ . The relative onset time bias is the difference of onset timing between the perfor-

mance and the score, divided by the total length of the phrase. Namely,

$$ROB = \frac{O_i^{perf} - O_i^{score}}{length(phrase)}$$



Where  $O_i^{perf}$  is the onset time of note  $i$  in the performance,  $O_i^{score}$  is the onset time of note  $i$  in the score.

However, the above formula assumes the performance is played at the tempo assigned in the score. In the corpus we use, test subject can't always keep up with the speed of the score because of limited piano skill, or they may speed up or slow down certain sections as their expression. Therefore, the performance should be linearly scaled to avoid such problem, We will discuss this issue in Section 3.5.3.

**Relative loudness:** The loudness of a note divided by the maximum loudness in the phrase. Measured by MIDI velocity level 0 through 127.


$$RL = \frac{L_i}{\max(L_1, L_2, \dots, L_n)}$$

**Relative duration:** The performed duration of note divided by the duration indicated in the score.

$$RD = \frac{D_i^{perf}}{D_i^{score}}$$

### 3.5.3 Normalizing Onset Deviation

In previous section, we mentioned that the onset deviation feature has problems when performance is not exactly the same length as the score. Illustrated in Fig. 3.11, if the performance is played faster than expected, the deviation will grow larger and larger over time. For a long phrase, the onset deviation of the last notes may be over a dozen quarter notes. These kind of extreme values will cause erroneous predictions in the model: a note may be delayed for a very large deviation, causing it to be played after the next note, the swapped notes will destroy the melody.



TBD			
Figure will be			
inserted here			

Figure 3.10: Example Performance Features

TBD			
Figure will be			
inserted here			

Figure 3.11: Problem with Onset Deviation

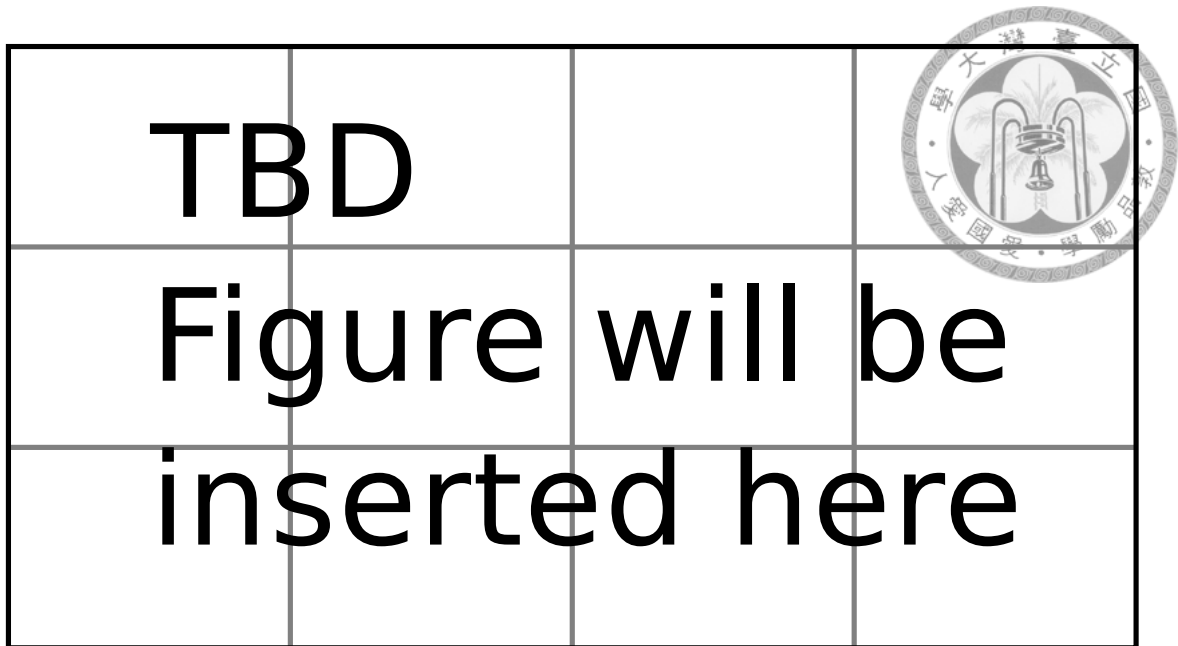


Figure 3.12: Normalization Schemes

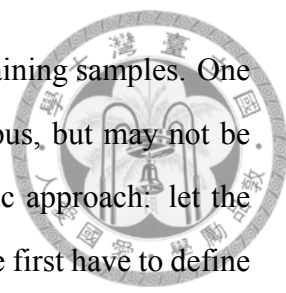
In other words, the original definition of the onset deviation actually contains two type of deviation: a global deviation cause by difference in the tempo, and a local deviation cause by note-level expression. Since the intention of the onset deviation features is to catch the note-level expression, the performance must be linearly scaled to cancel the global deviation.

Initially, we tried four possible type of normalization methods :

1. Align the onset of the first notes, align the onset of the last notes
2. Align the onset of the first notes, align the end of the last notes
3. Don't align the onset of the first notes, align the onset of the last notes
4. Don't align the onset of the first notes, align the end of the last notes

The incentive for not aligning the first note is that the performer may intend to use an early start or delayed start as an expression, if the first note is aligned by it's onset, the first note in every phrase will have a onset timing bias feature of value zero. In other words, the early/delayed start expression is lost.





But empirical data shows that none of these methods can fit all training samples. One method may produce good result when trained with part of the corpus, but may not be suitable for the rest of the corpus. So we switch to a more dynamic approach: let the program find the best ratio for normalization. To achieve this goal, we first have to define how “fit” two phrases are. If the onset time of all the notes in a phrase are concatenated into a vector, the  $l^2$ -norm of the two vectors can be treated as the distance. Note that the two vectors must have the same size, because the recordings are required to match note-to-note with the score. So the problem becomes how to find an optimal scaling ratio such that the scaled recording has the minimum distance from the score. The Brent's Method [?] is used to find the optimal ratio. To speed up the optimization and prevent unreasonable value, a search range of  $[initial\_guess \times 0.5, initial\_guess \times 2]$  is imposed on the optimizer. The *initial\_guess* is used as a rough estimate of the ratio, calculated by aligning the first and last onset of the phrase. Then we assume the actual ratio will not be smaller than half of *initial\_guess* and not larger than twice of *initial\_guess*. The two numbers 0.5 and 2 are arbitrary chosen, but most of the empirical data suggest is valid most of the time.

So we have defined an automatic method to dynamically adjust the normalization ratio to eliminate systematical error in the onset deviation feature. Comparing this method to not using any normalization (see Fig. 3.13), the method can produce very low onset deviations, roughly centered around zero, while the result from not using any normalization will show a clear trend of increasing deviations. ]



TBD			
Figure will be			
inserted here			

Figure 3.13: The Effect of Automatic Normalization on Onset Deviation feature



## Chapter 4

# Corpus Preparation

Since this research is based on a supervised learning algorithm, a high-quality corpus is essential to our success.

A expressive performance corpus is a set of performance samples. Each sample consists of a score and a recording. The score is the music notation being played, plus some metadata such as structure analysis, harmonic analysis etc.; the recording is a recording of a human musician playing the said score. A learning algorithm can learn how the music notation is transformed into real performance. In this chapter, we will review the existing corpora, sample specifications and formats of a corpus, and how we construct the corpus used in this research.

### 4.1 Existing Corpora

Unlike research fields like speech processing or natural language processing, there exist very few public accessible corpus for research. CrestMusePEDB [?] (PEDB stands for “Performance Expression Database”), created by Japan Science and Technology Agency's CREST program, is a rare example. It claims to contain the following data: PEDB-SCR - score text information, PEDB-DEV - performance deviation data and PEDB-IDX - audio performance credit. The database is said to be free to use via email request, but until the time of this writing, we can't establish any contact with the database administrators, so the quality and format of the data is unknown.

Another example is the Magaloff Project [?], which is a joint effort from a few universities in Austria. Russian pianist Nikita Magaloff was invited to record all works for solo piano by Frederic Chopin on a Bösendorfer SE computer-controlled grand piano. This corpus became the material for many subsequent researches [?]. Flossmann et al., the leading researchers of the project, also won the 2008 RenCon contest with a expressive performance system call YQX [?] based on this corpus. However, the corpus is not opened up in the public domain.



## 4.2 Corpus Specification

Since we can't find any public corpus for our experiment, we need to implement our own one. First, we need a clear specification of what will be included and will not.

The corpus we need must fulfill the following constraints:

1. All the samples are monophonic, containing only a single melody without chords.
2. No human error, such as insertion, deletion, or wrong pitch exist in the recording; the score and recording are matched note-to-note.
3. Phrasing information is given by human.
4. The score, recording and phrasing data are in machine-readable format.

There are many other useful information that can be included, but they are less relevant to our system, so they are not included. Examples are:

1. Detailed Structural Analysis, such as GTTM (Generative Theory of Tonal Music) [?]
2. Harmonic Analysis
3. Musical Instrument specific instructions, such as violin pizzicato, tapping, or bow techniques.
4. Musical Instrument specific instructions, such as piano fingering, violin bow techniques etc.

Table 4.1: Clementi's Sonatinas Op. 36

Title	Movement	Time Signature
No. 1 Sonatina in C major	I. Allegro	4/4
	II. Andante	3/4
	III. Vivace	3/8
No. 2 Sonatina in G major	I. Allegretto	2/4
	II. Allegretto	3/4
	III. Allegro	3/8
No. 3 Sonatina in C major	I. Spiritoso	4/4
	II. Un poco adagio	2/2
	III. Allegro	2/4
No. 4 Sonatina in F major	I. Con spirito	3/4
	II. Andante con espressione	2/4
	III. Rondó: Allegro vivace	2/4
No. 5 Sonatina in G major	I. Presto	2/2
	II. Allegretto moderato	3/8
	III. Rondó: Allegro molto	2/4
No. 6 Sonatina in D major	I. Allegro con spirito	4/4
	I. Allegro con spirito	6/8



## 5. Piano paddle usage

Clementi's Sonatina Op. 36 is chosen as the repertoire of the corpus. Because Clementi's Sonatina is a basic repertoire almost every piano student in Asia will learn, so it's easy to find performers with different skill level to record the corpus. Clementi wrote these sonatinas in classical style, so the skill required to play them can be easily extended to other classical era works like Mozart or Haydn. This fact makes the learned model a very general one, which is good for evaluation. There are six sonatinas included in Op. 36, the first five have three movements each, and the last one has two movements. The titles, tempo markers and time signatures of all the pieces in Op. 36 are listed in Table 4.1

There are many digital formats for score to choose from, such as MusicXML [52], LilyPond [53], Finale, Sibelius, ABC, MuseData, and Humdrum. The book [54] has a comprehensive review on this issue. We choose MusicXML as our score format. MusicXML is a score notation system using XML (eXtensible Markup Language) representation, it can express most music notations and metadata, and its specification is publicly available, so most music notation software and computer music code library will support musicXML format. An example snippet of a musicXML score is shown in Fig. 4.1 Although MIDI is

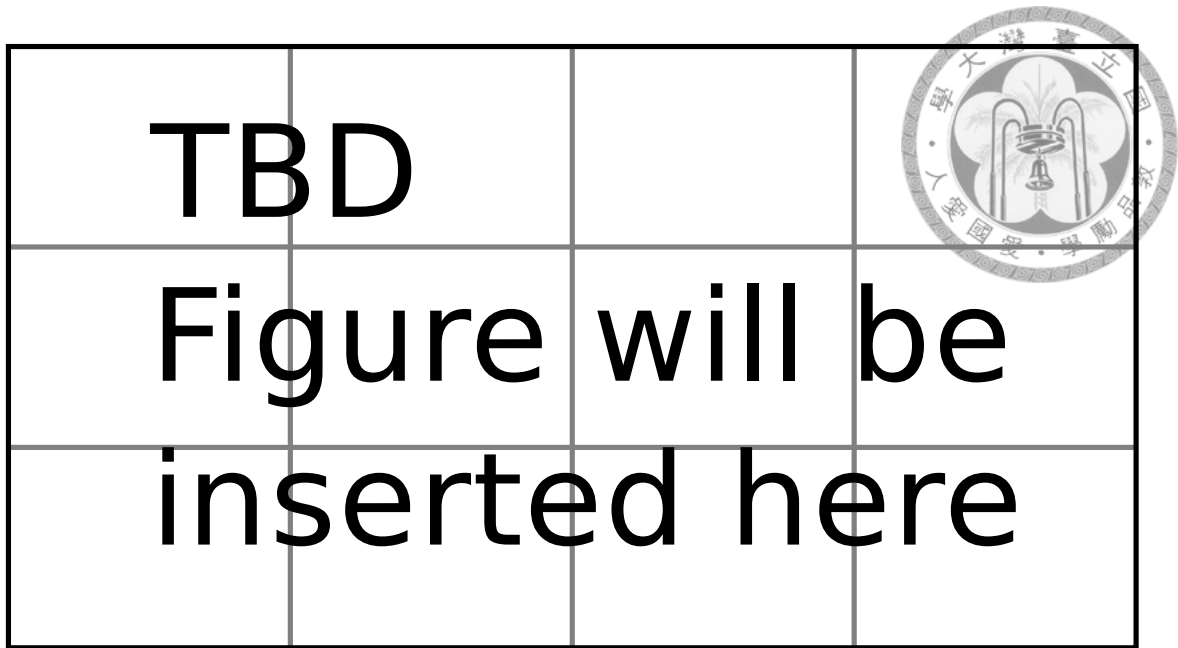


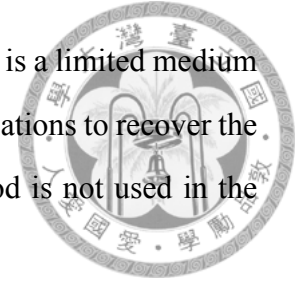
Figure 4.1: Example MusicXML score

also a popular candidate for score representation in computer music research, it is designed to hold instrument control signal rather than notation. Some printed music notations may not be available in MIDI. Furthermore, MIDI represents music as a series of note on and off events, which in nature does not fit the mental model of traditional music notation system.

But for performance, MIDI is the most suitable format. Although an WAV (Waveform Audio Format) audio recording has higher fidelity than MIDI, it takes extra effort to annotate the notes, either manually or automatically. Since onset detection and pitch detection algorithms still can't achieve perfect accuracy, manually labeling is inevitable, which will soon become an impossible if manpower is not sufficient. On the other hand, a MIDI recording can provide exact timing, key pressure, and pitch for each note, even in polyphonic recordings.

There's one possible way to keep the score and recording in one single MIDI file. Instead of recording the exact note on and note off timing, the nominal note on and off time are kept on the beats, which is exactly the same on the score. Then, MIDI tempo-change events are inserted before each note to represent the real timing of the recorded

notes. This way we can merge the two files into one. But since MIDI is a limited medium for score, as discussed in early section, and it requires complex calculations to recover the performance from fixed notes and tempo-change events, this method is not used in the research.



Finally, metadata not in the score and performance need to be stored somewhere. The only metadata we used is the phrasing. We store the phrasing in a plaintext file, each line in the phrasing file is the starting point of each phrase. The starting point is defined as the onset timing (in quarter notes) from the starting of the piece<sup>1</sup> The phrasing is assigned by the author, but since phrasing controls the structural expression of a piece, anyone can create their own phrasing file to make the system learn their phrasing decision. For our corpus, the boundaries of phrases are defined by the following features:

1. Salient pause.
2. Cadence
3. Dramatic change in tempo, key or loudness
4. Repeated structures in tempo or pitch.

The above are just general principles, not strict rules, the phrasing is still decided by subjective analysis.

## 4.3 Implementation

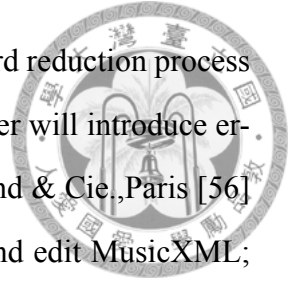
### 4.3.1 Score Preparation

The digital score used is downloaded from KernScore website [55]. The original format is in Hundrum file format (.krn), they are transformed into MusicXML by music21 toolkit. Because this research focus on monophonic melody only, the accompaniments are remove and the chords are reduced to their highest pitched note, which is usually the

---

<sup>1</sup>For a phrase that start at a point which is a circulating decimal, the starting point can be defined as any finite decimal between the end of the last phrase and the start of the current phrase. For example, if the last phrase stops at beat 1, the second phrase start at  $2\frac{1}{3}$  beat, the start point of the second phrase can be written as 2.3 or 2.0, etc.

most salient melody line. Since the accompaniment removal and chord reduction process is done by automated scripts, the bugs in the related musicXML parser will introduce errors. So the output is compared to a printed version publish by Durand & Cie., Paris [56] to eliminate any error. We use MuseScore notation editor to view and edit MusicXML; some metadata errors are corrected by editing the MusicXML with text editors .



### 4.3.2 MIDI Recording

We have implemented two methods to record an expressive performance: First, using a Yamaha digital piano to record MIDI. Second, by tapping on a laptop computer touchpad to express tempo, duration and loudness. Due to data accuracy consideration, only recordings from Yamaha digital piano are selected.

To record the MIDI performances, we used a Yamaha P80 88-key graded hammer effect<sup>2</sup>digital piano. The Yamaha keyboard was connected to a MIDI-to-USB converter so it acted as a USB MIDI device on a Linux computer. On the Linux computer we use Rosegarden Digital Audio Workstation (DAW) to record MIDIs. The Rosegarden DAW also generated the metronome sound to help the performer maintain a steady speed. One may argue that the tempo variation is also a part of the expression, but if the performer plays freely, the tempo information written in the MIDI file will be invalid, which makes subsequent parsing and manipulation a very hard task. So the performers are asked to follow the speed of the metronome, but they can apply any level of rubato they like.

TODO: touch pad recordings The second method, which is not used in the final experiments, is utilizing the Synaptics Touchpad on a Lenovo X200i laptop. When the user tap the touchpad, one note from the score will be played and recorded, when the user taps again, the next note will be played. The timing and pressure of the tapping event will be translated to the note's onset, duration and loudness. This idea have already be used in musical toys [?] and musical games. If this input mechanism is turned into a game, we can easily collect large quantity of training input from laptops and smartphones with touch-screen. But the problem with this method is the lack of accuracy in pressure, because the

---

<sup>2</sup>Graded Hammer Effect feature provides realistic key pressure response similar to a traditional acoustic piano



touchpad use the touched surface area to estimate the pressure. So we did not use samples from this method, but in early experiments we did find this method a plausible alternative to MIDI keyboard.



### 4.3.3 MIDI Cleaning and Phrase Splitting

After the MIDI files are recorded, a utility script is employed to check each recording is matched note-to-note with its corresponding score; if not, the mistakes are manually corrected using MIDI editing software. For example, if the pitch was played wrongly, we will correct the pitch but keep the onset, duration and intensity as is. If there are small segments that are beyond simple fix, repeated or similar segments from the same piece are used as a reference to reconstruct the wrongly-performed segment. The matched score and MIDI pair are then split into phrases according to the corresponding phrasing file using a script. The splitted phrases are checked again for note-to-note match to avoid bugs in the scripts.

## 4.4 Results

Clementi's Sonatina Op. 36 has six pieces, the number of phrases (according to our phrasing) and notes are shown in Table 4.2. The length distribution of the phrases in six pieces are shown in Fig. 4.2 TODO: distribution of corpus Six graduate students (not majored in music) with a varying piano skill. Their piano-related experiences are shown in Table 4.3. Five of them finished Clementi's entire Op. 36, the last one only recorded part of the work. The total number of recordings and the corresponding phrases/notes counts are shown in Table 4.4.

BOOKMARK

TODO: mention Fig. 4.3



Table 4.2: Phrases and Notes Count for Clementi's Sonatina Op. 36

Title	Phrases Count	Notes Count
No. 1 Mov. I	12	222
No. 1 Mov. II	10	147
No. 1 Mov. III	16	261
No. 2 Mov. I	18	320
No. 2 Mov. II	6	125
No. 2 Mov. III	28	414
No. 3 Mov. I	25	526
No. 3 Mov. II	6	74
No. 3 Mov. III	19	438
No. 4 Mov. I	25	465
No. 4 Mov. II	12	222
No. 4 Mov. III	16	384
No. 5 Mov. I	17	672
No. 5 Mov. II	13	316
No. 5 Mov. III	24	564
No. 6 Mov. I	28	836
No. 6 Mov. II	11	459
<b>Total</b>	<b>286</b>	<b>6445</b>

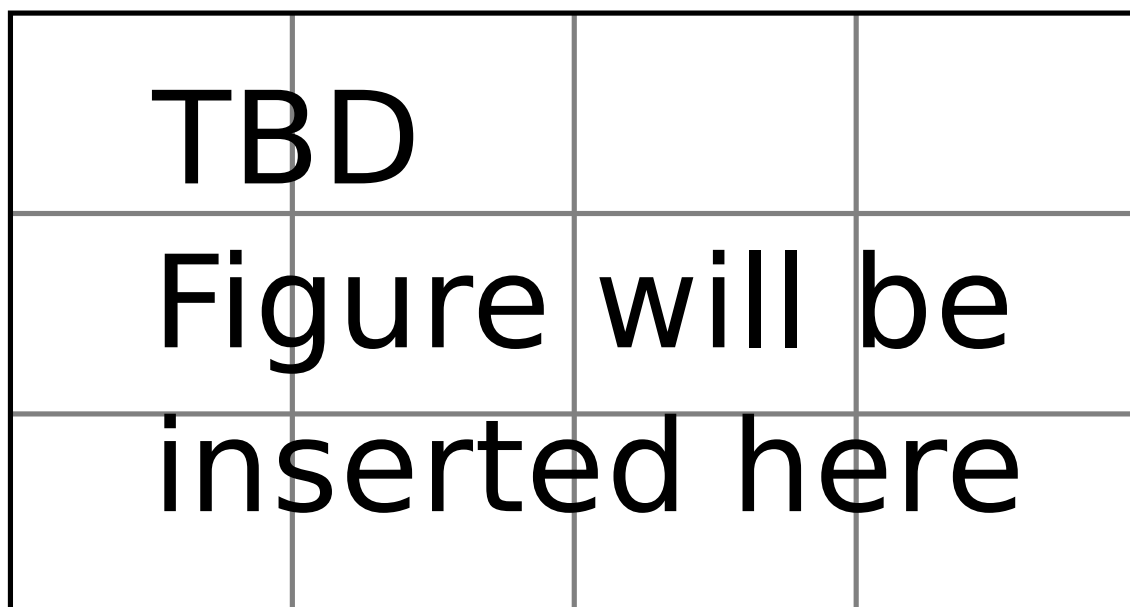


Figure 4.2: Phrase Length Distribution

Table 4.3: Performer Music Experience

TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD



Table 4.4: Total Recorded Phrases and Notes Count

TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD
TBD	TBD	TBD

TBD			
Figure will be			
inserted here			

Figure 4.3: Example Recording Compared to Score (Pianoroll)



## Chapter 5

# Experiments, Results and Discussion

TODO:general introduction to the experiments

### 5.1 Parameter Selection

#### 5.1.1 Experiment Design

Structural Support Vector Machine has some parameters that needed to be adjusted. We will leave the others to the defaults and change the SVM C trad-off parameter in this experiment. Since three models are learned for three performance features, we have three parameters to adjust.

[TODO: phrases count] phrases from [TODO:song counts] songs are used for training. Every first, fifth, and tenth phrases from each song is not included in the training sample, but used as testing samples. A three-by-three grid is layed out for three C parameters, each C takes the value of the powers of tenfrom [TODO: Cs]  $10^{-5}$  to  $10^4$ , so [TODO: num of experiment] paramenters are tested. Then the result is validated

1. Are all the output samples successfully generated? (Generation may fail if the performance features are unreasonable, for example, negative onset timeing.)
2. Is the order of the notes preserved? Sometimes the first note is delayed too long and the second note is played too early, so the order is swaped.

3. Are there any extreme parameters that makes the expressive performance unnatural?

The first two criterias are checked by python scripts, the last one is done by manual inspection.



## 5.1.2 Results and Discussion

TODO:experiment result

## 5.2 Human-like Performance

TODO:experiment result

### 5.2.1 Experiment Design

TODO:experiment result

### 5.2.2 Results and Discussion

TODO:experiment result

## 5.3 Performer Style Reproduction

TODO:experiment result

### 5.3.1 Experiment Design

TODO:experiment result

### 5.3.2 Results and Discussion

TODO:experiment result

## 5.4 Comparing with State-of-the-Art Works

TODO:experiment result



### 5.4.1 Experiment Design

TODO:experiment result

### 5.4.2 Results and Discussion

TODO:experiment result



## Chapter 6

## Conclusions

TODO:summary


TODO:futurework

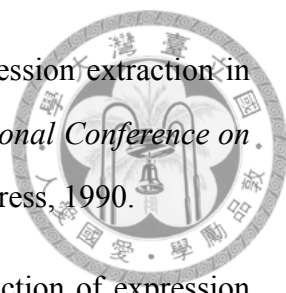


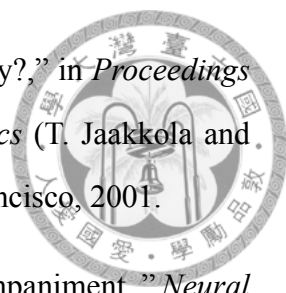
## Bibliography

- [1] A. Kirke and E. R. Miranda, "An Overview of Computer Systems for Expressive Music Performance," in *Guide to Computing for Expressive Music Performance* (A. Kirke and E. R. Miranda, eds.), pp. 1--47, Springer, 2013.
- [2] R. Hiraga, R. Bresin, K. Hirata, and R. KH, "Turing test for musical expression proceedings of international conference on new interfaces for musical expression," in *Proceedings of 2004 new interfaces for musical expression conference* (Y. Nagashima and M. Lyons, eds.), (Hamatsu, Japan), pp. 120--123, ACM Press, 2004.
- [3] "Rachmianinoff - Plays Rachmaninoff." [https:// www.zenph.com/ rachmaninoff-plays-rachmaninoff](https://www.zenph.com/rachmaninoff-plays-rachmaninoff), 2009.
- [4] "Sibelius." <http://www.avid.com/us/products/sibelius/pc/Play-perform-and-share>.
- [5] A. Friberg, R. Bresin, and J. Sundberg, "Overview of the KTH rule system for musical performance," *Advances in Cognitive Psychology*, vol. 2, pp. 145--161, Jan. 2006.
- [6] W. A. Sethares, *Tuning, Timbre, Spectrum, Scale*. Springer, 2005.
- [7] M. Hashida, N. Nagata, and H. Katayose, "Pop-E: a performance rendering system for the ensemble music that considered group expression," in *Proceedings of 9th International Conference on Music Perception and Cognition* (M. Baroni, R. Addressi, R. Caterina, and M. Costa, eds.), (Bologna, Spain), pp. 526--534, ICMPC, 2006.



- 
- [8] S. R. Livingstone, R. Mühlberger, A. R. Brown, and A. Loch, “Controlling musical emotionality: an affective computational architecture for influencing musical emotions,” *Digital Creativity*, vol. 18, pp. 43--53, Mar. 2007.
  - [9] N. P. M. Todd, “A computational model of rubato,” *Contemporary Music Review*, vol. 3, pp. 69--88, Jan. 1989.
  - [10] N. P. McAngus Todd, “The dynamics of dynamics: A model of musical expression,” *The Journal of the Acoustical Society of America*, vol. 91, p. 3540, June 1992.
  - [11] N. P. M. Todd, “The kinematics of musical expression,” *The Journal of the Acoustical Society of America*, vol. 97, p. 1940, Mar. 1995.
  - [12] M. Clynes, “Generative principles of musical thought: Integration of microstructure with structure,” *Journal For The Integrated Study Of Artificial Intelligence*, 1986.
  - [13] M. Clynes, “Microstructural musical linguistics: composers' pulses are liked most by the best musicians,” *Cognition*, 1995.
  - [14] M. Johnson, “Toward an expert system for expressive musical performance,” *Computer*, vol. 24, pp. 30--34, July 1991.
  - [15] R. B. Dannenberg and I. Derenyi, “Combining instrument and performance models for high-quality music synthesis,” *Journal of New Music Research*, vol. 27, pp. 211--238, Sept. 1998.
  - [16] R. B. Dannenberg, H. Pellerin, and I. Derenyi, “A Study of Trumpet Envelopes,” in *Proceedings of the 1998 international computer music conference* (O. 1998, ed.), (Ann Arbor, Michigan), pp. 57--61, International Computer Music Association, 1998.
  - [17] G. Mazzola and O. Zahorka, “Tempo curves revisited: Hierarchies of performance fields,” *Computer Music Journal*, vol. 18, no. 1, pp. 40--52, 1994.
  - [18] G. Mazzola, *The Topos of Music: Geometric Logic of Concepts, Theory, and Performance*. Basel/Boston: Birkhäuser, 2002.


- 
- [19] H. Katayose, T. Fukuoka, K. Takami, and S. Inokuchi, "Expression extraction in virtuoso music performances," in *Proceedings of 10th International Conference on Pattern Recognition*, vol. i, pp. 780--784, IEEE Comput. Soc. Press, 1990.
- [20] H. Katayose, T. Fukuoka, K. Takami, and S. Inokuchi, "Extraction of expression parameters with multiple regression analysis," *Journal of Information Processing Society of Japan*, no. 38, pp. 1473--1481, 1997.
- [21] O. Ishikawa, Y. Aono, H. Katayose, and S. Inokuchi, "Extraction of Musical Performance Rules Using a Modified Algorithm of Multiple Regression Analysis," in *International Computer Music Conference Proceedings*, (Berlin, Germany), pp. 348--351, International Computer Music Association, San Francisco, 2000.
- [22] S. Canazza, G. De Poli, C. Drioli, A. Rodà, and A. Vidolin, "Audio Morphing Different Expressive Intentions for Multimedia Systems," *IEEE MultiMedia*, vol. 7, pp. 79--83, July 2000.
- [23] S. Canazza, A. Vidolin, G. De Poli, C. Drioli, and A. Rodà, "Expressive Morphing for Interactive Performance of Musical Scores," p. 116, Nov. 2001.
- [24] S. Canazza, G. De Poli, A. Rodà, and A. Vidolin, "An Abstract Control Space for Communication of Sensory Expressive Intentions in Music Performance," *Journal of New Music Research*, vol. 32, pp. 281--294, Sept. 2003.
- [25] R. Bresin, "Artificial neural networks based models for automatic performance of musical scores," *Journal of New Music Research*, vol. 27, pp. 239--270, Sept. 1998.
- [26] A. Camurri, R. Dillon, and A. Saron, "An experiment on analysis and synthesis of musical expressivity," in *Proceedings of 13th colloquium on musical informatics*, (L'Aquila, Italy), 2000.
- [27] G. Grindlay, *Modeling expressive musical performance with Hidden Markov Models*. Phd thesis, University of Santa Cruz, CA, 2005.

- 
- [28] C. Raphael, “Can the computer learn to play music expressively?,” in *Proceedings of the 8th Int. Workshop on Artificial Intelligence and Statistics* (T. Jaakkola and T. Richardson, eds.), pp. 113--120, Morgan Kaufmann, San Francisco, 2001.
- [29] C. Raphael, “A Bayesian Network for Real-Time Musical Accompaniment.,” *Neural Information Processing Systems*, no. 14, pp. 1433--1440, 2001.
- [30] C. Raphael, “Orchestra in a box: A system for real-time musical accompaniment,” in *Proceedings of 2003 International Joint conference on Artificial Intelligence (Working Notes of IJCAI-03 Rencon Workshop)* (G. Gottob and T. Walsh, eds.), (Acapulco, Mexico), pp. 5--10, Morgan Kaufmann, San Francisco, 2003.
- [31] L. Dorard, D. Hardoon, and J. Shawe-Taylor, “Can style be learned? A machine learning approach towards ‘performing’ as famous pianists.,” in *Proceedings of the Music, Brain and Cognition Workshop -- Neural Information Processing Systems*, Whistler, Canada, 2007.
- [32] M. Wright and E. Berdahl, “Towards machine learning of expressive microtiming in Brazilian drumming,” in *Proceedings of the 2006 International Computer Music Conference* (I. Zannos, ed.), (New Orleans, USA), pp. 572--575, ICMA, San Francisco, 2006.
- [33] R. Ramirez and A. Hazan, “Modeling Expressive Music Performance in Jazz.,” in *Proceedings of 18th international Florida Artificial Intelligence Research Society Sonference (AI in Music and Art)*, (Clearwater Beach, FL, USA), pp. 86--91, AAAI Press, Menlo Park, 2005.
- [34] R. Ramirez and A. Hazan, “Inducing a generative expressive performance model using a sequential-covering genetic algorithm,” in *Proceedings of 2007 annual conference on Genetic and evolutionary computation*, (London, UK), ACM Press, New York, 2007.
- [35] Q. Zhang and E. Miranda, “Towards an evolution model of expressive music performance,” in *Proceedings of the 6th International Conference on Intelligent Systems*

*Design and Applications* (Y. Chen and A. Abraham, eds.), (Jinan, China), pp. 1189-1194, IEEE Computer Society, Washington, DC, 2006.



- [36] E. Miranda, A. Kirke, and Q. Zhang, “Artificial evolution of expressive performance of music: An imitative multi-agent systems approach,” *Computer Music Journal*, vol. 34, no. 1, pp. 80--96, 2010.
- [37] Q. Zhang and E. R. Miranda, “Evolving Expressive Music Performance through Interaction of Artificial Agent Performers,” in *Proceedings of ECAL 2007 workshop on music and artificial life (MusicAL 2007)*, (Lisbon, Portugal), 2007.
- [38] J. L. Arcos, R. L. De Mántaras, and X. Serra, “X. Serra, 1997. “SaxEx: a case-based reasoning system for generating expressive musical performances” ,” in *Proceedings of 1997 International Computer Music Conference* (P. Cook, ed.), (Thessalonikia, Greece), pp. 329--336, ICMA, San Francisco, 1997.
- [39] J. L. Arcos, R. L. De Mántaras, and X. Serra, “Saxex: A case-based reasoning system for generating expressive musical performances,” *Journal of New Music Research*, vol. 27, no. 3, pp. 194--210, 1998.
- [40] J. L. Arcos and R. L. De Mántaras, “An Interactive Case-Based Reasoning Approach for Generating Expressive Music,” *Journal of Applied Intelligence*, vol. 14, pp. 115-129, Jan. 2001.
- [41] T. Suzuki, T. Tokunaga, and H. Tanaka, “A case based approach to the generation of musical expression,” in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, (Stockholm, Sweden), pp. 642--648, Morgan Kaufmann, San Francisco, 1999.
- [42] T. Suzuki, “Kagurame phase-II,” in *Proceedings of 2003 International Joint Conference on Artificial Intelligence (working Notes of RenCon Workshop)* (G. Gottlob and T. Walsh, eds.), (Acapulco, Mexico), Morgan Kaufmann, Los Altos, 2003.

- 
- [43] K. Hirata and R. Hiraga, “Ha-Hi-Hun: Performance rendering system of high controllability,” in *Proceedings of the ICAD 2002 Rencon Workshop on performance rendering systems*, (Kyoto, Japan), pp. 40--46, 2002.
- [44] G. Widmer, “Large-scale Induction of Expressive Performance Rules: First Quantitative Results,” in *Proceedings of the 2000 International Computer Music Conference* (I. Zannos, ed.), (Berlin, Germany), pp. 344--347, International Computer Music Association, San Francisco, 2000.
- [45] G. Widmer, “Discovering simple rules in complex data: A meta-learning algorithm and some surprising musical discoveries,” *Artificial Intelligence*, vol. 146, pp. 129--148, 2003.
- [46] G. Widmer and A. Tobudic, “Playing Mozart by Analogy: Learning Multi-level Timing and Dynamics Strategies,” *Journal of New Music Research*, vol. 32, pp. 259--268, Sept. 2003.
- [47] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, “Large Margin Methods for Structured and Interdependent Output Variables,” *Journal of Machine Learning Research*, vol. 6, pp. 1453--1484, 2005.
- [48] T. Joachims, T. Finley, and C.-N. J. Yu, “Cutting-plane training of structural SVMs,” *Machine Learning*, vol. 77, pp. 27--59, May 2009.
- [49] Y. Altun, I. Tsochantaridis, and T. Hofmann, “Hidden Markov Support Vector Machines,” in *Proceedings of the 20th International Conference on Machine Learning*, vol. 3, (Washington DC, USA), pp. 3--10, 2003.
- [50] S. H. Lyu and S.-k. Jeng, “COMPUTER EXPRESSIVE MUSIC PERFORMANCE BY PHRASE-WISE MODELING,” in *workshop on Computer Music and Audio Technology*, 2012.
- [51] T. Joachims, “SVM<sup>hmm</sup>: Sequence Tagging with Structural Support Vector Machines,” 2008.

- [52] M. Good, “MusicXML: An Internet-Friendly Format for Sheet Music,” in *XML Conference hosted by IDEAlliance*, 2001.
- [53] “LilyPond.” <http://www.lilypond.org>.
- [54] E. Selfridge-Field, *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, 1997.
- [55] “KernScores.” <http://kern.ccarh.org/>.
- [56] M. Clementi, *SONATINES pour Piano a 2 mains Op. 36 VOLUME I [Musical Score]*. Paris: Durand & Cie., plate d. & c. 9318 ed., 1915.
- [57] T. Joachims, T. Finley, and C. Yu, “Cutting-plane training of structural SVMs,” *Machine Learning*, vol. 77, pp. 27--59, May 2009.





## **Appendix A**

### **Software Tools Used in This Research**



## **Appendix B**

# **Using the Expressive Performance Corpus**