國立交通大學

資訊科學與工程研究所

博 士 論 文

縮編式自動編曲之研究

Automatic Music Arrangement by Score Reduction

研 究 生：邱士銓

指導教授：黃俊龍

共同指導教授：沈錳坤

中 華 民 國 一 百 年 五 月

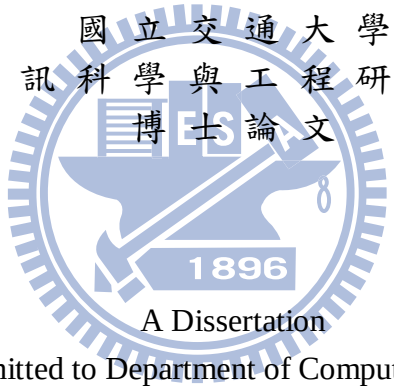
縮編式自動編曲之研究

Automatic Music Arrangement by Score Reduction

研 究 生：邱士銓　　Student：Shih-Chuan Chiu

指導教授：黃俊龍　　Advisor：Jiun-Long Huang

共同指導教授：沈錳坤　　Co-Advisor：Man-Kwan Shan



國立交通大學

資訊科學與工程研究所

博士論文

A Dissertation

Submitted to Department of Computer Science

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

May 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年五月

縮編式自動編曲之研究

學生：邱士銓　　指導教授：黃俊龍　博士

共同指導教授：沈錳坤　博士

國立交通大學　資訊科學與工程研究所

摘要

樂譜縮編(Score Reduction)是一個透過縮編樂譜來達成為單一樂器編曲的過程。在本篇論文中，我們提出了一個使用樂譜縮編方法的編曲架構，此架構可以自動化為一樣樂器進行編曲。根據樂譜縮編的方法，我們要盡可能地包含原曲的每一個部份，並同時滿足目標樂器可彈奏性的限制，使得編曲出來的音樂聽起來和原曲相同。在本架構的第一個步驟中，我們分析原始曲目的音樂編曲元素(Arrangement Element)。接著，將音樂中的每個樂句辨識出來，並且根據音樂編曲元素分析的結果與樂句的特性，來分配每個樂句的重要程度。最後，我們將音樂編曲轉換成一個最佳化的問題並設計一個演算法來解決這個問題。藉由挑選適當的樂句並且同時考量目標樂器的可彈奏性，來完成編曲。在實驗中，我們使用這個編曲架構來實作一個鋼琴編曲的系統。許多的實驗被設計來評估我們系統所產生出來的音樂。為了避免主觀的影響，我們採用了一個類似圖林測試(Turing Test)的方法來評估這個系統的好壞。實驗的結果證明我們的系統有能力編寫出具品質且可彈奏的曲子。

此外，為了捕捉到原始音樂的特色，我們介紹了一種新的樣式－多音重覆樣式，並提出兩個演算法－*A-PRPD* (Apriori-based Polyphonic Repeating Pattern Discovery) and *D-PRPD* (Depth-first-search based Polyphonic Repeating Pattern Discovery)，從原始音樂中探勘多音重覆樣式。再者，我們設計了位元方法(bit approach)用於我們所提出的兩個演

算法上加速運算。實驗結果顯示，我們提出的演算法是有效率與效果的，並且 *D-PRPD* 演算法加上位元方法在大多數的情況下是最有效率的演算法。此探勘出的重覆樣式可以被使用在此音樂編曲架構中的功能性分配的步驟中，使得具有原始音樂特色的樂句較容易被挑選到。

Automatic Music Arrangement by Score Reduction

Student: Shih-Chuan Chiu Advisor: Dr. Jiun-Long Huang

Co-Advisor: Dr. Man-Kwan Shan

Department of Computer Science

National Chiao Tung University

Abstract

Score reduction is a process that arranges music for a target instrument by reducing original music. In this dissertation we present a music arrangement framework that uses score reduction to automatically arrange music for a target instrument. According to the approach of score reduction, the goal is to include as many important parts of the original music as possible within the constraint of the target instrument so that the arranged version is similar to the original. In our proposed framework, the original music is first analyzed to determine the type of arrangement element of each section. Then, the phrases are identified and each is assigned a utility according to its type of arrangement element. For a set of utility-assigned phrases, we finally transform the music arrangement into an optimization problem and propose a phrase selection algorithm to solve it. The music is arranged by selecting appropriate phrases satisfying the playability constraints of a target instrument. Using the proposed framework, we implement a music arrangement system for the piano in our experiments. Several experiments were conducted to evaluate our system. To avoid subjective opinions, one approach of the experiments similar to Turing-test is used to evaluate the quality of the music arranged by our system. The experimental results show that our system is able to create viable music for the piano.

To capture the characteristics of the music for enhancing the proposed music arrangement

framework, we introduce a new type of repeating patterns, polyphonic repeating pattern and propose algorithms, *A-PRPD* (Apriori-based Polyphonic Repeating Pattern Discovery) and *D-PRPD* (Depth-first-search based Polyphonic Repeating Pattern Discovery), to discover them from music data. Furthermore, a bit-string approach is developed for improving the efficiency of both proposed algorithms. Experimental results show that the proposed algorithms are both effective and efficient for mining polyphonic repeating patterns from synthetic music data and real data, and *D-PRPD* with bit-string approach is the most efficient approach in most cases. The discovered polyphonic repeating patterns can be used to enhance in the phrase identification and utility assignment phase of our proposed framework such that the phrases with music characteristics will be easy to be selected.

Acknowledgement

(誌謝)

本論文與博士學位得以順利完成，首先得感謝我的兩位指導教授－黃俊龍老師，以及沈錳坤老師在研究過程中的悉心指導。在這些年的研究所生涯中，有黃老師一路的提攜與指導，使得研究能力上得以打穩基礎，並且在研究方法、研究討論、論文修改與投稿上給我極大的協助；另一方面，政大沈老師在研究上的執著與熱忱啟發我對學術研究的興趣，並且教導我很多教學與論文報告上的技巧。跟著兩位老師學習讓我受益良多，在此謹致上我最誠摯的感謝。此外，還要感謝校內的口試委員李素瑛老師與彭文志老師在計劃書口試、校內口試與校外口試時提供許多寶貴的意見。

同時也感謝校外的口試委員台大電機陳銘憲教授、台大資管李瑞庭教授與清大資工張智星教授對論文細心審閱，以及在口試過程中提供許多寶貴的建議，讓本篇論文的內容能夠更加充實、完善。諸位口試委員都是我在學術研究道路上的最佳學習典範。

感謝交大行動與普及計算實驗室(MPC-Lab)與政大多媒體與資料探勘實驗室(DM-Lab)的學弟妹，謝謝你們在這段時間給我的幫忙與鼓勵，祝福學弟妹們在事業或學業上都有很好的成就。

最後感謝我的父母與妹妹給我的愛與關懷，讓我無後顧之憂。還有一直陪在我身邊的睦美，讓我能夠用正面、樂觀的態度來面對這個世界，能夠順利地完成博士學位。

謹以此論文，獻給我摯愛的太太睦美。

邱士銓　謹誌予

交通大學資訊科學與工程研究所

中華民國一百年五月

Table of Contents

List of Tables

List of Figures

CHAPTER 1 INTRODUCTION

"Over the Rainbow," a classical ballad has remained popular since 1939. As of now, there are more than 100 versions of this song, interpreted by numerous artists using different organizations of instruments in various styles. For example, Jason Castro sang it in reggae style, accompanied by a ukulele; jazz artists, Tommy Emmanuel used his guitar; and Robert Kyle played a monophonic tenor sax. When a song is to be performed by an instrument or an ensemble, a process called music arrangement or transcription is necessary to adapt the song for the target instrument(s) [15]. Music arrangement gives existing melodies more variety.

In the music industry, there are many applications of music arrangement. For example, although the average mobile phone now doubles as music player, the function of the customized ring tone still appeals to people. Music arrangement transforms the original music object into various styles. There is another issue regarding mobile phones: the problem of transcoding from MIDI to SP-MIDI (specific polyphonic MIDI) [37]. Due to hardware limitations, most mobile phones support only SP-MIDI. The polyphony has to be reduced and its impact on the music, minimized. Music arrangement that reduces multipart instruments can achieve the same goal. However, the process of extracting the essential part from the original music is always time-consuming for the arranger. Besides, not every music arranger is familiar with the properties of the particular instrument. Thus, we believe that automatic music arrangement is needed to address the problems stated above.

Generally, there are two major approaches to arranging music. One is rewriting a piece of existing music with additional material. Instead of adding new material, the other one is score reduction that arrangers reduces the original work from a larger score to a smaller score. That is, the arranger does not create new counterpoints, harmonies, bass lines, and voices, but only focuses on eliminating the less important parts of the original score for application to the

target instrument and keeps the arranged version similar to the original. Piano reduction is a word which specifically refers to a two-line staff of a basic component reduced from multipart music for a piano. Many famous piano reductions include the Bach transcriptions of Concerto from various composers (bwv 972-987), Wagner/Liszt Tannhäuser, the Sullivan transcriptions of Concerto Violoncello and orchestra, and Sheherazade Op. 35 of Nikolai Andreyevich Rimsky-Korsakov [45]. In this dissertation, we concentrate on score reduction for two reasons. First, score reduction allows a musician to perform a musical piece using the instrument with which he/she is familiar. Second, less prior studies on the literature focus on how to automatically create an instrument-playable arrangement.

When arranging a piece of music for a target instrument, it is necessary to take the characteristics of original music and the inherent restrictions of the target instrument, such as pitch range and polyphonic limitation. Simply speaking, the goal is to include as many parts of the original music as possible within the constraint of the target instrument so that the arranged version is similar to the original. In addition, the role of an instrument varies in the different organizations of an ensemble. For example, in a big band, the guitar may play accompaniment; however, for a solo, it may perform melody and accompaniment simultaneously. The arrangement for the different roles of an instrument needs to be considered. To achieve this, we apply the concept of *arrangement elements* to take into account the different roles of an instrument. The type of arrangement element of a piece of music presents the function performed by an instrument in the piece of music. According to the book [42], there are five types of arrangement elements: *lead*, *foundation*, *rhythm*, *pad* and *fill*. Interested reader can be referred to the book [42] for more discussions about the arrangement elements. To summarize, there are three factors need to be considered: 1. the role of the target instrument; 2. the characteristics of the original music; and 3. the playability.

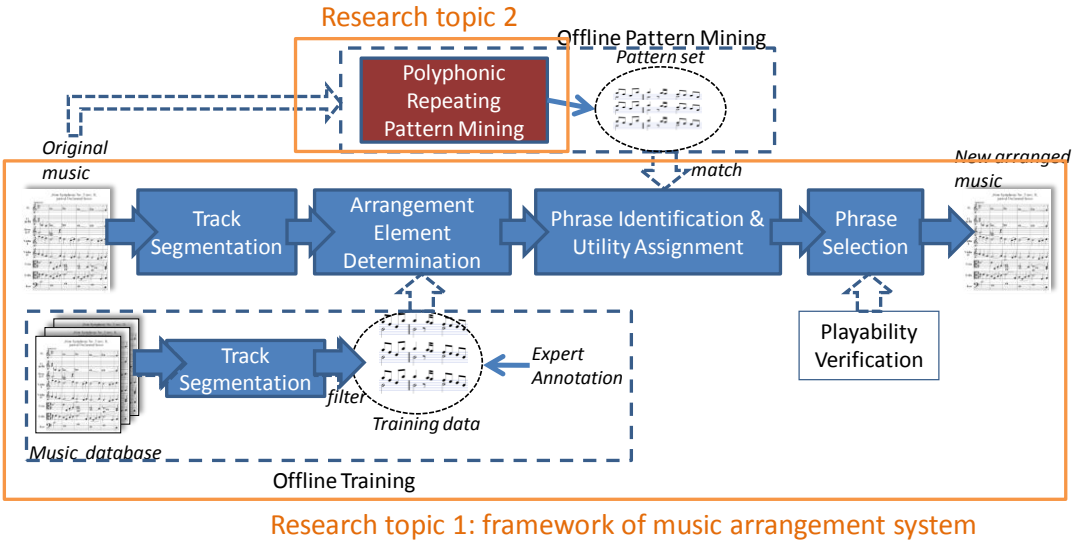


Figure 1-1. The flowchart of the proposed music arrangement framework

In this dissertation, we propose a framework that arranges a piece of music by reducing the multipart score for a given instrument as our first research topic. The main characteristic of the framework is that the various roles of the target instrument in an ensemble can be specified by users. Given an original score (multipart) and the role of the target instrument (proportion of the five types of arrangement elements), the proposed framework will generate a playable arrangement for the target instrument according to the role user specified. The framework consists of four phases (see Figure 1-1). First, the music object is first divided into several segments in *track segmentation* phase. Next, in the *arrangement element determination* phase, a classifier is used to determine the type of arrangement element for each instrument. The classifier is trained offline by expert-annotated tracks. In the *phrase identification and utility assignment* phase, the phrases in a segmented track are identified, and the utility is assigned for each identified phrase according to the type of arrangement element of the segmented track. In the *playability verification* phase, a playability verification function is used to determine whether the given piece of music can be played by the target instrument. Finally, in the *phrase selection* phase, the phrases are selected according to their utility and playability. The new arranged music is formed by these selected phrases. Based on

the proposed framework, we implement a music arrangement system for the piano. Several experiments are conducted to evaluate the system.

To capture the characteristics of the music, a new type of repeating patterns, polyphonic repeating pattern, is investigated as our second research topic. We propose two algorithms, *A-PRPD* (Apriori-based Polyphonic Repeating Pattern Discovery) and *D-PRPD* (Depth-first-search based Polyphonic Repeating Pattern Discovery) to discover them from music data. Furthermore, a bit-string approach is developed for improving the efficiency of both proposed algorithms. Experimental results show that the proposed algorithms are both effective and efficient for mining polyphonic repeating patterns from synthetic music data and real data, and *D-PRPD* with bit-string approach is the most efficient approach in most cases. The discovered polyphonic repeating patterns can be used to enhance in the phrase identification and utility assignment phase of our proposed framework. For example, while the identified phrases are similar to the discovered repeating patterns, the utility of the identified phrases can be increased.

The remainder of this dissertation is organized as follows. Chapter 2 gives a preliminary of this dissertation including the related work of automatic music arrangement and polyphonic repeating pattern mining. Chapter 3 gives an introduction of each component of the proposed music arrangement framework. The other research work of polyphonic repeating pattern mining is discussed in Chapter 4. The experimental results are given in Chapter 5 while Chapter 6 concludes this dissertation.

CHAPTER 2 PRELIMINARY

2.1 Introduction of Arrangement Elements

In *The Mixing Engineer's Handbook*, Owsinski proposed a taxonomy — the so-called arrangement elements — for the function of a piece of music performed by an instrument [42]. Analyzing the arrangement elements will help musicians understand the structure of the arrangement so that they can do further processes on the music, such as arranging, mixing, etc. According to the book [42], there are five types of arrangement elements: *lead*, *foundation*, *rhythm*, *pad*, and *fill*.

Lead the melody and its counterpoint. The melody is the clearest part of music that people usually remember and hum. The *lead* is usually demonstrated by a lead vocal or solo instrument.

Foundation the main rhythm in music. It is always a regular pattern played by a drum (especially bass drum and snare) or bass instrument.

Rhythm broken bits counted to the *foundation* played by any instrument. It is more complicated in beat and used to increase music fluency.

Pad consists of a long sustaining note or chord. Hence, it is usually played by a string instrument, organ, or synthesizer. Generally, the *pad* can also denote those sounds which create ambiance.

Fill usually appears in the spaces between the *lead* lines to fill up the silence between successive phrases of *lead*. It is similar to conversation: If the *lead* is a call, the *fill* would be a response.

These five elements can be viewed as the ingredients of an arrangement. The role of an

instrument can be referred to as an arrangement element or a mixture of them. In this dissertation, "role" and "arrangement element" are used interchangeably.

A passage played by an instrument can be considered to have the property of one or more arrangement elements. Roughly speaking, in an arrangement, the instrument is played for presenting the role of melody or accompaniment, or both of them for a solo. If it presents a melody, the proportion of the *lead* is especially higher than the others. If it presents accompaniment, the situation is reversed. For a solo with melody and accompaniment, the distribution is more uniform. In depth, beyond two rough roles, the subtle role can also be described on the distribution over these five elements. For example, when many instruments play accompaniment in music, some focusing on *pad* and some on *rhythm*, these subtle roles of the different distributions can be showed. By understanding the arrangement elements of music passages, it will be useful to arrange for the various roles of an instrument in music.

2.2 Related Work of Automatic Music Arrangement

Many works related to music arrangement focus on how to transform original music by changing meta-information (tempo, timbre, etc.) or content (insert note, change pitch, re-assemble music segments, etc.) [39]. Nagashima and Kawashima employed chaotic neural networks to create variations on melodies [41]. The examples of the variations of an original music object are sent to train chaotic neural networks. The networks model the characteristics of the variations and make a new variation of the original music. Berndt presented the strategies to synchronize and adopt the game music with player interaction behaviors [5]. The approach to arrange music in the context of the interaction of applications is to vary the rhythmic, harmonic, and melodic elements of the basic theme. Chung proposed a real-time music-arranging system that reacts to the affective cues from a listener [12]. The system re-assembles a set of music segments according to the inferred affective state of a listener.

Based on a probabilistic state transition model, the target affective state can also be induced.

As to the reduction technique of score reduction for an instrument, piano reduction is one of the important terms particularly referred to a two-line staff of piano reduced from multipart music. Finale, a commercial software for music notation (http://www.finalemusic.com), provides a plug-in tool: piano reduction that combines a previously-prepared score into a two-line staff separated by a user-defined pitch value. However, due to the direct combination of notes in the score without selection, the part of produced score may be difficult or even impossible to play. Finale's tool just provides a platform on which arrangers can do further piano reduction. Since the research on guitar fingering became mature [50], Daniel et al. presented an approach for guitar arrangement [16]. The main concept is to choose a set of important notes by a search algorithm, with the constraint on the playability of the guitar. However, this approach is dedicated to a solo guitar and cannot arrange for various roles in music. In addition, we argue that if the chosen notes came from different instruments, it may result in the loss of musical meaning, such as the completeness of a piece of melody.

2.3 Related Work of Polyphonic Repeating Pattern Mining

The repeating pattern mining problem has been investigated in the last decade. The first method to solve this problem is to utilize a suffix tree to find repeating patterns in a DNA sequences [46]. Suffix tree is a well-known data structure originally developed for string matching. Repeating patterns can be extracted from a suffix tree, which is constructed by sharing common prefix of a string. Since there may exist a large number of repeating patterns in a sequence, the concept of non-trivial repeating pattern was introduced [33]. Hsu et al. proposed two approaches to efficiently find the repeating patterns in a music object [18][18]. In the first approach, a data structure called correlative matrix is constructed to keep the intermediate information for substring matching. The lengths and frequencies of all repeating

patterns can be derived according to the information in this matrix. The other approach, called string-join, utilized the anti-monotony property to avoid generating large amount of candidate. Here the anti-monotonic property is that if a string is frequent, then all its substrings are frequent. According to this property, shorter frequent patterns are joined into longer ones and the non-qualified candidates are pruned out. Since a suffix tree was able to construct in linear time [38], Lo et al. employed this improvement to find repeating patterns [34], [36].

Since some minor variances in the instances of a repeating pattern are tolerable in some areas such as music, many approaches are investigated to find approximate repeating patterns. Two previous mentioned approaches, string-join and correlative matrix, are modified [35] to find approximate repeating patterns. The distance between a pattern and its occurrence is defined by edit distance. A novel approach treating pattern discovery as instance search problem is proposed [32]. This method segments a string into a set of small pieces and maps these pieces into a multi-dimension space based to search in the multi-dimension space to count the number of occurrences. Two techniques are incorporated to improve the process. In addition, in bio-informatics, some approximate algorithms are designed to take advantage of the special properties in DNA strings, such the few kinds of items [46] and short non-tandem patterns [1]. Many different types of repeating patterns are proposed to accommodate to varied patterns in music [25][11]. In music field, Lartillot proposes a series of work on discovering musical patterns [27][28][29]. The main idea of these methods is that each pattern is induced by analyzing the music sequence in chronological order. This process is similar to the simulation of listening strategy of human. All possible combinations of successive events are stored and checked. Therefore, very high computation and storage costs are required to deal with a longer sequence. The approaches mentioned above model music data as a melody line (a string) and find the repeating patterns on the string. That is, they are designed to find *monophonic* patterns from *monophonic* music.

Some work focuses on finding *monophonic* patterns from *polyphonic* music. They define the special type of repeating pattern in polyphonic music, such as vertical patterns and perceptible repetitions. Conklin analyzes the vertical patterns, which is common harmonic progress, from one or more music objects by encoding it or them into a set of strings [13][14]. The experimental result shows that most of vertical patterns represent specific voice leading formulae within cadences[1]. Meudic discovers the perceptible repetitions from audio [40]. The process first segments a music object. Then, the similarity between each pair of segmentations is computed according to perception they defined. Finally, the perceptible repetitions are discovered from the similarity matrix. A geometrical pattern proposed by Meredith et al is represented in polyphonic form [39]. Many significant patterns occurring more than two times can be found. However, certain meaningful patterns, such as motif, usually appear several times in music, and it takes time to use the result of pair-wise repeats to count the number of occurrences of a pattern.

[1] A cadence is a piece of music ends a section of music or a complete piece of music.

CHAPTER 3 AUTOMATIC MUSIC ARRANGEMENT FRAMEWORK

In this chapter, we introduce our proposed framework. Given an original score (multipart) and the role of the target instrument (proportion of the five types of arrangement elements), the proposed framework will generate a playable arrangement for the target instrument according to the role user specified. The framework consists of four phases (see Figure 1-1). Figure 3-1 presents an example of the change in musical content during the framework's arrangement process. Because the arrangement elements of some instruments may change in different sections, the music object is first divided into several segments called *segmented tracks* in track segmentation phase. Next, in the arrangement element determination phase, a classifier is used to determine the type of arrangement element of each segmented track. The classifier is trained offline by expert-annotated segmented tracks. In the phrase identification and utility assignment phase, the phrases in a segmented track are identified, and the utility is assigned for each identified phrase according to the type of arrangement element of the segmented track. In the playability verification phase, a playability verification function is used to determine whether the given piece of music can be played by the target instrument. Finally, in the phrase selection phase, the phrases are selected according to their utility and playability. The new arranged music is formed by these selected phrases.

The details of each phase of the proposed music arrangement framework are introduced in the following subsections. A list of notations used in this paper is shown in Table 3-1 for better readability.

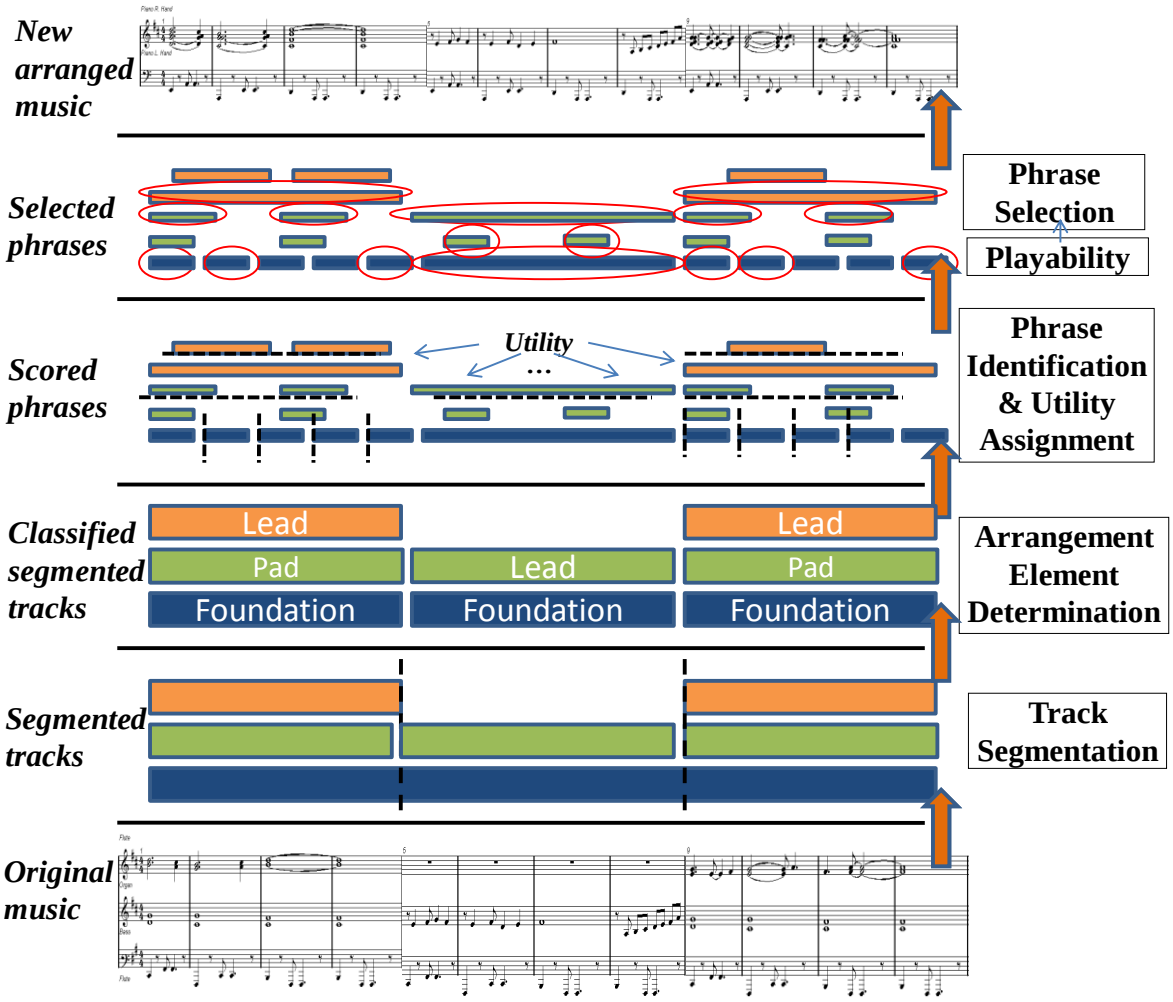


Figure 3-1. An example process of the proposed music arrangement framework

Table 3-1. List of used notations

Notation	Description
$NSBM_{i,t}$	Non-silent beats in measure *i* at track *t*
NumTrack	Number of track
BeatPerMeasure	Beats per measure
ae	Arrangement element
st	Segmented track
phr	Phrase
room	Number of overlapping phrase allowed
MOP	Maximum number of overlapping phrase allowed
P_List	Overlapping phrase list
C_List	Called phrases in list
SP	Set of selected phrase

3.1 Track Segmentation Phase

In different sections, a track performed by an instrument may belong to different types of arrangement elements. For example, a violin demonstrating *pad* arrangement element changes to *lead* in the violin solo section. Hence, the track is segmented into *segmented tracks*. A segmented track is defined as a period of an instrument's performance in which no arrangement element changes. Here we do not analyze musical sections; instead, we want to ensure that no arrangement element changes in a segmented track. Since the multipart music usually possesses a more complete arrangement structure, we apply this benefit in solving the problem. In other words, a time point, where many instruments stop and others start, has a high possibility of becoming a cut point to separate two adjacent segmented tracks. According to this heuristic, we define the similarity function between consecutive measures as follows.

$$Sim_{i,i+1} = 1 - \left(\frac{\sum_{t \in Track} \left| NSBM_{i,t} - NSBM_{i+1,t} \right|}{NumTrack \times BeatPerMeasure} \right) \tag{1}$$

where $NSBM_{i,t}$ is the number of non-silence beats in measure i at track t, *NumTrack* is the number of tracks, and *BeatPerMeasure* is beats per measure.

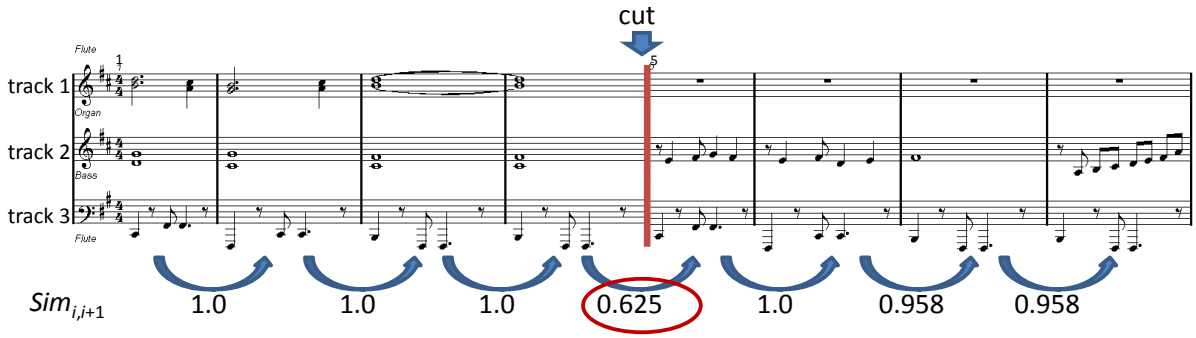


Figure 3-2. An example of track segmentation

The similarity function compares the track in measure i to the track in measure (i+1), then aggregates diversities of all tracks with normalization. Being subtracted by 1, the difference is transformed into similarity. We define a threshold value τ to determine cut points. If $Sim_{i,i+1}$ is

less than τ, then this is a cut point between measures i and i+1. When τ is set to 0.5, $Sim_{i,i+1}$ < τ, it means that there must be at least a half number of instruments switched.

Figure 3-2 gives an example that shows the similarity for each pair of successive measures in this music. For the similarity between measure 4 and 5, $Sim_{4,5}$ is calculated as follows. In the example music, time signature is 4/4 with 3 tracks; that is, each measure has 4 beats (*BeatPerMeasure*=4, *NumTrack*=3). As the score of the track 1 shown, sound fulfills four beats in measure 4 ($NSBM_{4,1}$=4), and it is no sound in measure 5 ($NSBM_{5,1}$=0). Similarly, the others ($NSBM_{4,2}$, $NSBM_{5,2}$, $NSBM_{4,3}$, $NSBM_{5,3}$) can be derived. Thus, $Sim_{4,5}$=1－((|4－0|＋|4－3.5|＋|3－3|)/(4×3))=0.625. If τ is set to 0.667, there is a cut point between measure 4 and 5.

3.2 Arrangement Element Determination Phase

Here we try to determine the type of the arrangement element of each segmented track. According to the descriptions of the arrangement elements in Chapter 2, some arrangement elements share similar properties. It is hard to determine the type of the arrangement element by heuristic rules. Hence we treat the problem of arrangement element determination as a classification problem. In other words, each segmented track is classified into five classes (i.e., *foundation*, *rhythm*, *pad*, *lead*, and *fill*).

One of the important steps of classification is to decide which features are used to represent the segmented track. These features of a segmented track are capable of discriminating its class from the others. Most of previous studies on music classification focus on music style; to the best of our knowledge, there is no study in the literature about the automatic classification of arrangement elements.

Table 3-2. Features for the classifier

Parameter	**Type**	**Description**
AvgPitch	G	Average pitch in the segmented track
AvgDuration	G	Average duration in the segmented track
DevPitch	G	Pitch deviation in the segmented track
IsPercussionChannel	G	Is Percussion Channel (usually channel 10)
PolyphonicRate	G	Proportion of note occurring in the same time
SilentRate	G	Proportion of silent in the segmented track
AvgPitchRank	L	Rank of average pitch in parallel segmented track
AvgDurationRank	L	Rank of average duration in parallel segmented track
IsHighestPitchPart	L	Is the segmented track with the highest average pitch in parallel segmented tracks
IsLowestPitchPart	L	Is the segmented track with the lowest average pitch in parallel segmented tracks

G: global feature, L: local feature

According to the descriptions of the arrangement elements in the book [42], we summarize their characteristics and choose the features accordingly. The properties of an instrument exert a heavy influence on the arrangement element; for example, pizzicato instruments (such as harp, ukulele, etc) cannot be *pad*. The arrangement element of a segmented track highly depends on the others in this music, especially parallel ones. Thus, we choose both global feature (common features) and local features (related to the other segmented track). The detailed features that we extracted and their descriptions are listed in Table 3-2.

The classifier is trained using manual tagged data for each segmented track, i.e., a segmented track is marked as one of five types of arrangement elements according to its features. During the determination process, each segmented track in the given music is fed into the classifier to determine the type of arrangement element. The probability distribution over five types of arrangement element is obtained in our framework for the later phase.

In implementing of the arrangement element determination, we chose the support vector machine (SVM) [6] as our classifier. The SVM is a supervised learning approach. Input data is viewed as two sets of vectors in an *n*-dimensional space. In the space, the SVM constructs a separating hyperplane which maximizes the margin between the two data sets. A good

separation is achieved while the hyperplane has the largest distance to the neighboring data points of both classes. After the hyperplane is decided (training phase), the SVM model is able to answer or predict the class of a new example.

The sequential minimal optimization algorithm is employed for training a support vector classifier using the polynomial kernel. There are five classes in the arrangement element determination problem. The multi-class result can be solved by using pairwise classification; that is, the result is from C_2^m binary classifiers. Besides, the probability that a segmented track belongs to each class is vital information for our system. To obtain proper probabilities, logistic regression models are used to fit to the outputs of the support vector machine. In the multi-class case, Hastie and Tibshirani's pairwise coupling method [17] is employed with the predicted probabilities. It will input test data (a segmented track) to the classifier, then the probability distribution will be obtained as important information for utility assignment.

3.3 Phrase Identification and Utility Assignment Phase

3.3.1 Phrase Identification

In this subsection, we attempt to identify the phrases from a segmented track. As mentioned in [49], the definition of "phrase" is ambiguous. The phrase we try to find is a monophonic melodic group of notes with similar properties, usually separated by a breathe point or a large pitch interval. Many approaches have been proposed, which have performed well in finding this type of phrases. Because the phrases are found from a monophonic piece of music, we first have to identify the monophonic piece lines from a segmented track. Thus, the process of phrase identification consists of two steps: (1) finding monophonic lines; and (2) identifying phrases from monophonic lines.

In the first step, we adopt the approach proposed by Lui [37] because, to the best of our knowledge, no other studies on this topic have investigated so far. One of the most important issues of finding the monophonic line in polyphonic music is to preserve the best voice leading, which keeps the most natural melodic continuity between notes. The notes are grouped as follows: First, the chord progress of each measure is determined. For each consecutive pair of chords, let C_{fewer} be the chord with fewer notes and C_{more} be the chord with more notes. Resolve each tendency tone, and then each note of C_{fewer} is grouped with its neighbor of the nearest pitch in C_{more}. For different chords, the notes are grouped based on the following:

- For common chords, such as I and V, use *voice-leading matrixes* to resolve tendency notes.
- For the other chords, group each note of the preceding chord with its nearest neighbor in the succeeding chord.

The voice-leading matrix is two-dimensional (12×12). The indices are relative to the tonic and the entry indicates the voice leading priority from pitch row to pitch column. Interested readers can refer to [37] for the detailed descriptions.

In the first step, the monophonic lines are extracted. In the second step, the phrases are identified in each monophonic line. We investigated many works on this issue, and chose, the local boundary detection model (LBDM) [8] due to its easy implementation and good performance. The approach identifies phrases by segmenting a monophonic line according to larger pitch intervals or breaths of long notes. This model consists of a *change* rule, which assigns boundary strengths in proportion to the degree of change between consecutive intervals, and a *proximity* rule, which scales the boundary strength according to the size of the intervals involved. The LBDM performs over three independent parametric melodic profiles

$Profile_k = [x_1, x_2, \ldots, x_n]$ where $k \in \{ pitch, ioi, rest \}$, $i \in \{1, 2, \ldots, n\}$ and *ioi* stands for inter-onset interval. The boundary strength at interval x_i is defined by

$$strength_i = x_i \times (r_{i-1,i} + r_{i,i+1}) \tag{2}$$

where $r_{i-1,i}$ is the degree of change between two successive intervals and can be calculated by

$$r_{i,i+1} = \begin{cases} \dfrac{|x_i - x_{i+1}|}{x_i + x_{i+1}} & if\ x_i + x_{i+1} \neq 0 \wedge x_i + x_{i+1} \geq 0 \\ 0 & if\ x_i = x_{i+1} = 0 \end{cases} \tag{3}$$

For each parameter *k*, the boundary strength profile $strength_i$ is calculated and normalized into the range [0, 1]. A weighted sum of strengths is computed, using weights derived by trial-and-error in the previous study [8] (0.25 for *pitch* and *rest*, and 0.5 for *ioi*). Finally, the boundaries are detected where the combined strength profile exceeds a predefined threshold.

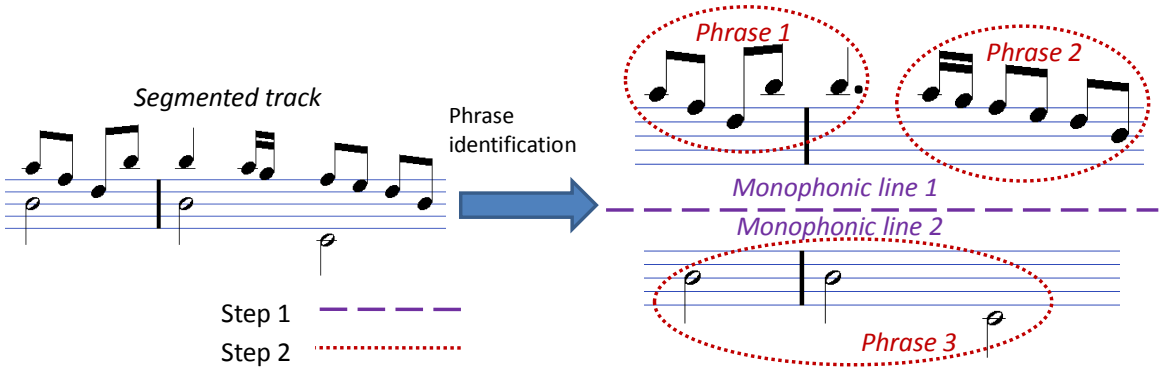


Figure 3-3. An example of phrase identification

Figure 3-3 illustrates an example of performing phrase identification. The given segmented track is polyphonic in left-hand side. In the step 1, the monophonic lines will be identified. In the beginning, A5 overlaps with B4, and two temporary monophonic lines, *tml*1 (A5) and *tml*2 (B4), are formed. It keeps grouping the notes, F5 and D5, for *tml*1 successively. Now *tml*1 contains A5, F5 and D5. When *tml*1 (A5, F5, D5) goes to B5, two notes, B5 and B4, can

be chosen. According to chord progress and pitch difference, B5 is grouped into *tm*1 and B4 is grouped into *tml*2. By the same process, monophonic line 1 and 2, *ml*1 and *ml*2, are formed. Then, *ml*1 and *ml*2 are fed into LBDM. When processing *ml*1, the cut point between the 5th and 6th note of *ml*1 is found because the combined strength profile exceed the threshold. Finally, three phrases are identified in the example.

3.3.2 Utility Assignment

Each of phrases identified is of different importance for the arrangement. We define the importance of a phrase, called *utility*, based on two factors. In the first factor, we consider the types of arrangement elements of the phrase for the target instrument that users considered. As mentioned in Section 3.2, the five types of arrangement elements in a segmented track have been determined and the classifier outputs the probabilities. Considering the input of our framework, the types of arrangement elements that users want to arrange for the target instrument have been specified in advance. The probabilities of the user-defined types of arrangement elements are taken as the first part of utility. Hence, the probabilities that the phrase inherited from the segmented track to which it belongs are summed up. To normalize the value, it is divided by the number of the considered types. The first factor, denoted as $F_1(phr_{st,i})$, can be formulated as

$$F_1(phr_{st,i}) = \sum^{ae} P(ae \mid st) \times \varphi_{ae} \Big/ \sum^{ae} \varphi_{ae} \tag{4}$$

where $phr_{st,i}$ is the *i*-th phrase in segmented track *st*; *ae* ∈{*Foundation*, *Rhythm*, *Pad*, *Lead*, *Fill*}; $P(ae|st)$ is the probability that the segment track *st* belongs to arrangement element *ae*; φ_{ae} is the user preference on arrangement element *ae* and $\varphi_{ae} \in (0, 1]$. For example, if we consider the arrangement elements, *lead* and *fill*, are important, we can set φ_{lead} and φ_{fill} to 1 and set the others close to 0. Note that for all phrases in the same segmented track, their F_1

values are equal.

In the second factor, the richness of a phrase is considered because we think it will make newly arranged music richer. The entropy is used to measure the richness of a phrase; that is, the phrase is richer when the pitches of the phrase are represented by more bits. The second factor, $F_2(phr_{st,i})$, is defined with the formula

$$F_2(phr_{st,i}) = normalize(-\sum_{i=1}^{m} pv_i \log_2(pv_i)) \tag{5}$$

where m is the number of distinct pitch values in the phrase $phr_{st,i}$ and pv_i is the proportion of a pitch value in a phrase.

Note that an upper bound for entropy is defined and the entropy can be normalized into 0 ~ 1. Here, the upper bound of the entropy is set to 64 heuristically, since a phrase usually falls within two measures and there are 16 distinct pitches at most for the notes with the 1/8 minimal length of a note in 4/4 music.

We combine the values of these two factors as the utility of a phrase with predefined weights. Since the phrases needed to be selected on score and some constraints exist among phrases over the time domain, the range of value leads into a situation wherein most of selected phrases are shorter. To assign the utility fairly over the time domain, the length of the phrase is also considered. Therefore, the utility of a phrase $U(phr_{st,i})$ is defined as

$$U(phr_{st,i}) = (\alpha_1 F_1 + \alpha_2 F_2) \times L(phr_{st,i}) \tag{6}$$

where $\alpha_1, \alpha_2 \in [0, 1]$; $\alpha_1+\alpha_2=1$; and $L(phr_{st,i})$ is the length of phrase $phr_{st,i}$.

3.4 Phrase Selection Phase

3.4.1 Phrase Selection Problem

After preparing the phrases with utilities, in the last phase of our framework, the phrases are selected under some conditions. Such selection is called the phrase selection problem and the formal definition of the phrase selection problem is as follows. For an arbitrary phrase *p*, its start position, end position, and utility over each arrangement element are denoted by *p.start*, *p.end*, and *p.utility*, respectively. *MOP* is an integer that denotes the maximal number of overlapping phrases, allowed by an instrument, simultaneously. Then, the phrase selection problem can be defined as below.

Definition 3-1 (Phrase selection problem) Given a set of phrases, denoted as $PSet=\{p_1, p_2,..., p_n\}$ and an integer *MOP*, the phrase selection problem is to find a set $SP \subseteq PSet$ such that:

1. the summation of the utilities of phrases in *SP* is maximal and

2. *SP* satisfies the constraints of *MOP* and playability.

The phrase selection problem is similar to the *k-track assignment problem*, which has been proved to be NP-hard, in the traditional job scheduling area [7]. The k-track assignment problem is a scheduling problem, in which a collection of jobs with start and end times is to be processed by k machines. Two different jobs can be processed by the same machine only when the jobs do not overlap. If the constraint of playability is omitted, the phrase selection problem will degenerate to the k-track assignment problem where k is equal to *MOP*. That is, the k-track assignment problem is a special case of the phrase selection problem. In addition to considering the constraint of the number of overlapping phrases (i.e., *MOP*), the phrase selection problem also needs to consider the playability of the selected phrases on the target

instrument. Thus, we believe that the phrase selection problem is more complex than the k-track assignment problem.

A naïve approach to solving the phrase selection problem is to integrate playability verification in to the algorithm [7] for the k-track assignment problem. Unfortunately, it is difficult to perform such integration since the algorithm proposed by Brucker and Nordmann [7] is optimized for the k-track assignment problem. Let's consider another problem, the exon chaining problem [21], which is a special case of the k-track assignment problem with k=1. Due to the simplicity of the algorithm proposed by Jones and Pevzner [21], we can extend such algorithm to consider playability verification and the scenarios with $k > 1$ simultaneously. For better readability, the descriptions and the design principle of playability verification are given in Section 3.4.2, while the proposed phrase selection algorithm is described in Section 0.

3.4.2 Playability Verification

In our proposed framework, we use the playability function to verify whether a piece of music can be performed by the instrument. The input of the playability function of an instrument is a piece of music and the output is a Boolean value indicating whether the music is playable by the instrument or not. Specifically speaking, the input is a set of phrases where the overlaps among the phrases may exist. The output value of a playability function can be determined by rules or sophisticate logic. We suggest some necessary considerations in designing a playability function as follows.

Playability Function Design Principle

To design the playability function of an instrument, two types of limitations have to be considered: instrumental and physical limitations. In instrumental limitation, we list some

constraints below.

Pitch range Pitch range is an important limitation for most instruments. For example, the pitch range of the piano is from the A three octaves below middle C to the C four octaves above middle C (if middle C is C4, it is A0~C8) [52]. The pitch range of a C flute is B3~C7.

Duration constraint Some instruments cannot sound sustain note, such as vibraphone.

Physical limitations are caused by hands or bodies of the people who play the instrument. We also list some constraints as follows.

Algorithm Piano-Right-Hand-Playability
Input: a piece of music (or a set of phrases *P_List*)
Output: True/False

```
1:  ons(note_i)={note|∀ note , note overlaps with note_i};
2:  nos_set={ons(note_i) | overlapping note sets in P_List};
3:  foreach note n in P_List{
4:    if pitch of note is not within the pitch range of piano
5:        return false;
6:  }
7:  foreach ons(note_i) in ons_set{
8:    if(Finger-Assignable(ons(note_i))==false)
9:        return false;
10: }
11: return true;
```

Figure 3-4. Piano-Right-Hand-Playability function

Number of polyphony Number of polyphony of an instrument is the maximal number of notes that the instrument can sound simultaneously. For example, people play the piano by right hands, so that at most, five notes can be played at the same time.

Physical pitch range constraint These constraints are caused by hand. The notes in the selected phrases are restricted by the expansion of the fingers.

Overlapping note constraint Some combinations of overlapping notes cannot be played. For example, B, C and C# cannot be played simultaneously by hand on the guitar.

Based on the design principle, we design a playability function for a right hand playing piano.

The playability function will also be used for the implementation of our piano arrangement system in the experiments.

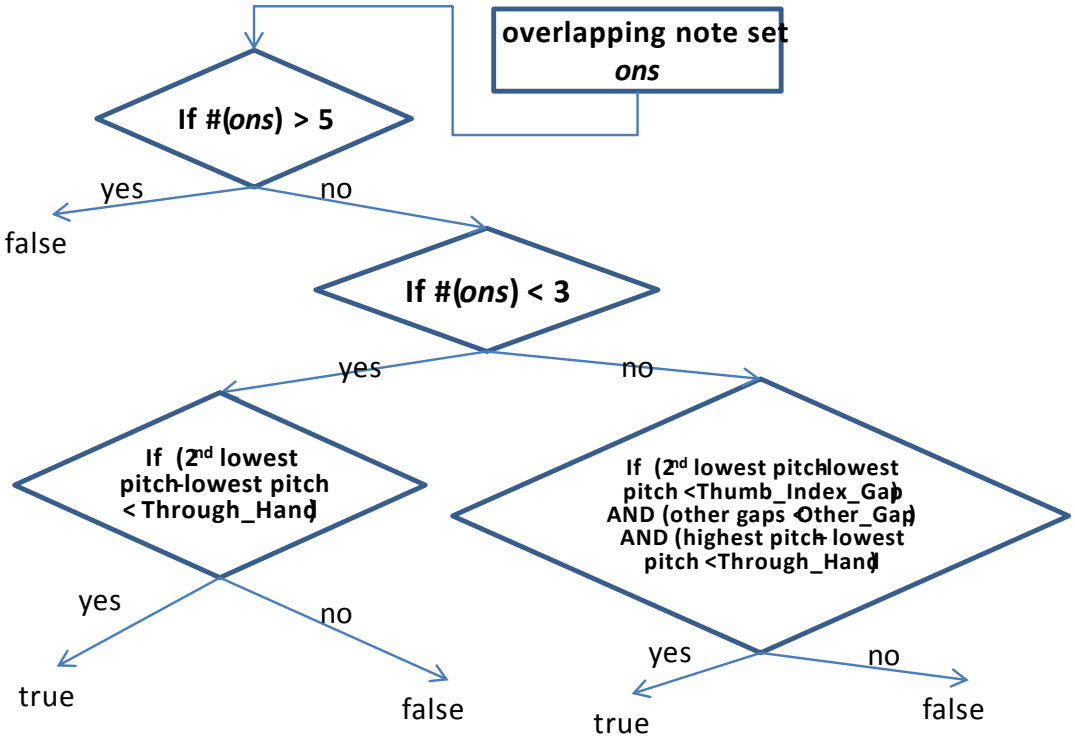


Figure 3-5. Finger-Assignable flowchart

Design of the Piano Playability Function

Here we design a playability function, *Piano-Right-Hand-Playability*, that considers instrumental and physical limitations for a right hand playing piano, as an example to illustrate the design of the playability functions. Research on automatic piano fingering has been investigated [23][24][54]; however, the work cannot be used to determine whether a piece of music can be played by piano. We refer to the book [52] to design this function. According to the phrase we defined, we assume that a single phrase is playable unless at least one note in the phrase is out of the pitch range of the instrument. The playability function for right hand is designed in Figure 3-4. The function is fed by a set of phrase, denoted by *P_List*,

and will output true or false to indicate whether these phrases can be played by the target instrument. First, the set of all overlapping note sets in *P_List*, denoted by *ons_set*, are extracted (lines 1-2). Note that the overlapping note set, $ons(note_i)$, is a set of notes overlapping with $note_i$ and $ons(note_i)$ includes at least one element, $note_i$. Two main rules are designed to examine the phrases and the phrases passing both rules are playable. The first rule (lines 3-6) checks each pitch of note to determine whether it is under the pitch range of piano. In the second rule (lines 7-10), we examine each overlapping note set in the phrase set to check whether it assignable for fingers of right hand by Finger-Assignment function.

In Figure 3-5, we give the flowchart of *Finger-Assignable* function. The number of notes in *nos* is examined first. If it is larger than five, then it is impossible to play by right hand and Finger-Assignment function will return false. If not, we will consider two cases: the case that the number of *nos* is two and the case that the number of *nos* is between 3~5. These cases are considered separately because the expansion of thumb-index finger is different from the other adjacent fingers. If the number of *ons* is two, we only have to ensure that the distance between the highest and the lowest notes does not exceed the distance between thumb and little finger, denoted by *Through_Hand*. Otherwise, while the number of *nos* is larger than two, the gap between thumb and index, denoted by *Thumb_Index_Gap*, can be larger than the gaps among the other fingers. We assume the legal gap distances among the other fingers are the same and all of them are denoted by a value, *Other_Gap*. That is, the distance between the lowest pitch and the second lowest pitch can be larger than the distance among others. According to the size of general fingers of an adult, we set these parameters heuristically: *Through_Hand*=14 semitones, *Thumb_Index_Gap*=5 semitones and *Other_Gap*=3 semitones. These parameters can be specified by users according to the size of their hands. Finally, the set of overlapping phrases is playable since all *ons* are assignable for fingers.

Algorithm: Phrase selection algorithm
Input: a set of phrases *PSet* and maximum overlapping phrase *MOP*
Output: selected phrase set *SP*

1: sort the start and end positions of all phrases;
2: initialize conditional phrase list *CP_List*=null;
3: extract 3 attributes for each index; // *phrase*, *utility* and *startI*
4: *SP*=*Opt*(0, 0, 2*n*－1, *MOP*| *CP_List*).*sel*;
5: **return** *SP*;

Algorithm *Opt*
Input: base value *bv*, start index *si*, end index *ei*, allowed overlapping phrases *room*, conditional phrase list *CP_List*
Output: selected phrase *sel* and utility of the selection *ut*

1: *Opt*(*bv*, *si*, *si*, *room*| *CP_List*).*ut*=*bv*;
2: *Opt*(*bv*, *si*, *si*, *room*| *CP_List*).*sel*=null;
3: **for each** index *i* from *si*+1 to *ei*
4: **if** ((*g*(*i*).*phrase*≠null **AND** *isPlayable*(*CP_List*∪{*g*(*i*).*phrase*} **AND** *room*＞0) **AND** //condition 1 (it is playable)
5: compute *w* by equation 7
6: (*w*＞*Opt*(*bv*, *si*, *i*−1, *room*| *CP_List*).*ut*) //condition 2 (it is worth to be selected)
7: *Opt*(*bv*, *si*, *i*, *room*| *CP_List*).*ut* = *w*; //update optimal utility and optimal selection by new result
8: update *Opt*(*bv*, *si*, *i*, *room*| *CP_List*).*sel*;
9: **else**
10: inherit optimal selection and utility from previous result (*Opt*(*bv*, *si*, *i*−1, *room*| *CP_List*));
11: **return** *Opt*(*bv*, *si*, *ei*, *room*| *CP_List*).*sel*, *Opt*(*bv*, *si*, *ei*, *room*| *CP_List*).*ut*;

Figure 3-6. Phrase selection algorithm

3.4.3 Phrase Selection Algorithm

The idea of the proposed phrase selection algorithm is to consider each phrase incrementally to determine whether it can be selected or not. To select a phrase, two conditions should be satisfied: 1. the phrase is playable with the previous selected phrases; 2. the phrase is worth to be selected. While checking whether a phrase is worth to be selected, we examine the influence of selecting the phrase on the previous selection. The whole problem can be divided into several small sub-problems.

Algorithm Overview

The details of the proposed phrase selection algorithm are shown in Figure 3-6. In the initialization step (lines 1-3), the placement of all phrases is transformed by sorting their start and end positions. The transformation will not change the order of the start and end positions of phrases, and will still keep the overlap relationships between each pair of phrases[2]. There are 2*n* indices for all start and end indices of *n* phrases. After that, for each index *i*, the following three attributes are extracted: *g*(*i*).*phrase*, *g*(*i*).*utility*, and *g*(*i*).*startI*. If index *i* corresponds to the end index of a phrase, *g*(*i*).*phrase* is the corresponding phrase, *g*(*i*).*utility* is the utility of *g*(*i*).*phrase* and *g*(*i*).*startI* is the start index of *g*(*i*).*phrase*. Otherwise, *g*(*i*).*phrase*, *g*(*i*).*utility*, and *g*(*i*).*startI* are null. A conditional phrase list, *CP_List*, is prepared to store a set of conditional phrases. Then, the main function, *Opt*(0,0,2*n*－1,*MOP*|{}), is called to compute the optimal selection *Opt*(0,0,2*n* － 1,*MOP*|{}).*sel* and the optimal utility *Opt*(0,0,2*n* － 1,*MOP*|{}).*ut* (the summation of the utilities of the selected phrases), where *MOP* indicates the maximal number of the overlapping phrases allowed by the target instrument. Finally, the proposed algorithm returns *SP* as the optimal selection. We define *Opt* as follows.

Definition 3-2 *Opt*(*bv*, *si*, *ei*, *room*| *CP_List*) is a function to compute the optimal selection of the phrases before index *ei* under the constraints that 1. the maximal number of overlapping phrases from index *si* to index *ei* is *room* and 2. the phrases in *CP_List* have been selected. The initial base value *bv* is the utility of the optimal selection of the phrases *seen* at *si*. A phrase is said to be seen at index *j* if the end position of the phrase is smaller than or equal to *j*. That is, *g*(*i*).*phrase* is said to be *seen* at index *j* if $i \leq j$.

[2] Interested readers can refer to [21] for the details of the transformation.

Function *Opt*

The most important part of the phrase selection algorithm is function *Opt*. To facilitate the following discussion, the utility of the selected phrases is defined as the summation of the utilities of these selected phrases. The objective of the function is to obtain the optimal selection and the utility of the optimal selection. The whole problem can be divided into several sub-problems, recursively. The process of function *Opt* is to sequentially check each phrase according to its end position in ascending order and determine whether the checked phrase is selected or not.

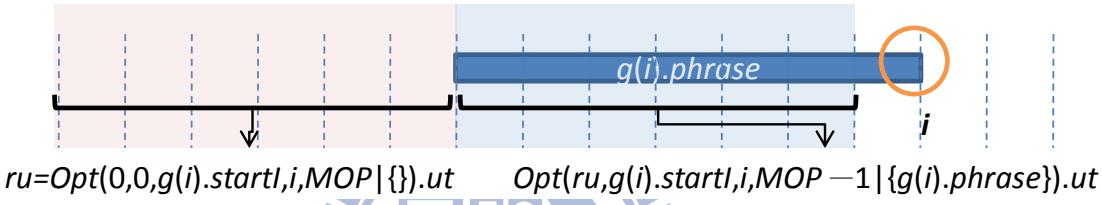


Figure 3-7. An illustration of *CP_List*={}

A phrase is selected only when the following two conditions are satisfied. The first condition is the *playable* condition (line 4) that *CP_List* ∪{*g*(*i*).*phrase*} should be playable and there is enough space for selecting *g*(*i*).*phrase*. Note that *g*(*i*).*phrase* cannot be null. The expression of the first condition is (*g*(*i*).*phrase*≠null and (*isPlayable*(*CP_List* ∪{*g*(*i*).*phrase*})=true and *room* > 0). The other condition is the *worth* condition (line 6) that the optimal utility of selecting *g*(*i*).*phrase* is worthier than not selecting *g*(*i*).*phrase*. That is, *w* > *Opt*(*bv*, *si*, *i*−1, *room*| *CP_List*), where the calculation of *w* will be described later. If the above two conditions are satisfied, *g*(*i*).*phrase* is selected. The optimal selection *Opt*(*bv*, *si*, *i*, *room*| *CP_List*).*sel* is updated according to the optimal selection during computing *w*, and the optimal utility *Opt*(*bv*, *si*, *i*, *room*| *CP_List*).*ut* is set to *w*. Otherwise, the optimal selection and the utility of the optimal selection are inherited from the previous results, *Opt*(*bv*, *si*, *i*−1, *room*| *CP_List*).*sel* and *Opt*(*bv*, *si*, *i*−1, *room*| *CP_List*).*ut*, respectively.

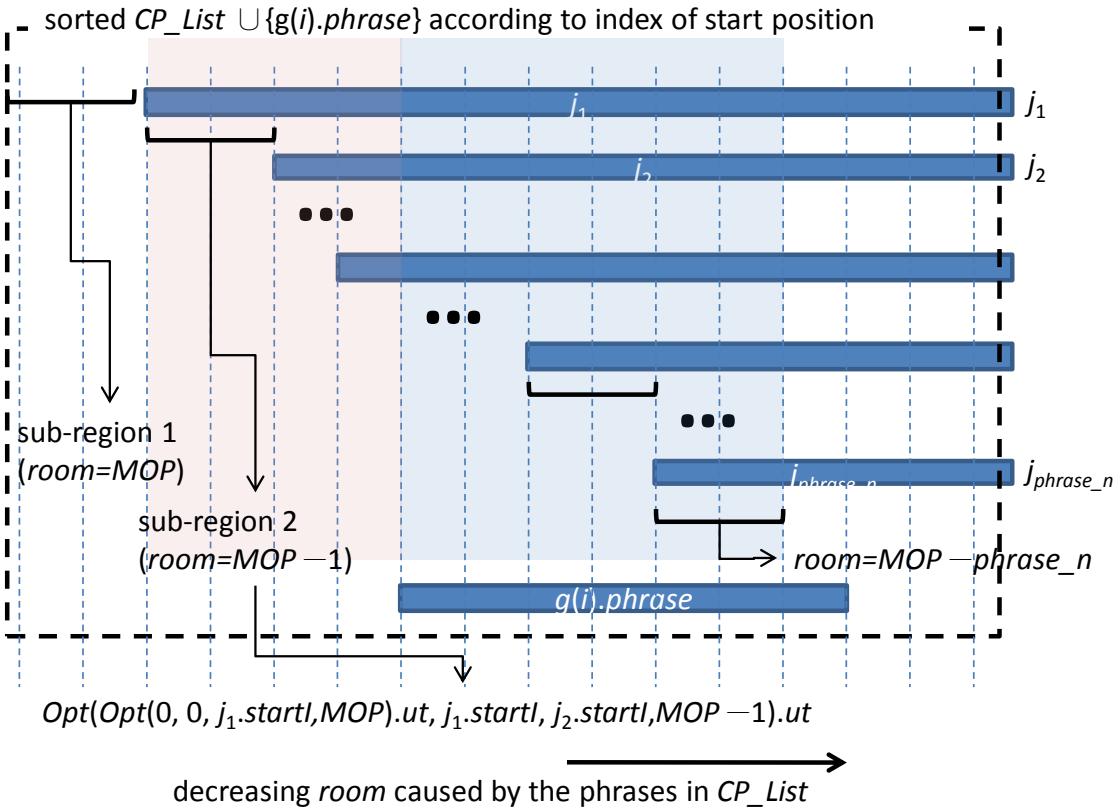


Figure 3-8. An illustration of *CP_List*≠{}

CP_List is empty

We now consider that *g*(*i*).*phrase* is selected for computing *w*. Note that selecting a new phrase may influence the optimal selection. That is, some phrases in the optimal selection may be removed due to the selection of the new phrase. Let's begin from the simple case that *CP_List* is empty. As shown in Figure 3-7, the influence region of selecting *g*(*i*).*phrase* is the region that *g*(*i*).*phrase* locates, i.e., from index *g*(*i*).*startI* to *i*. In addition, the maximal number of overlapping phrases allowed in the influence region of selecting *g*(*i*).*phrase* would be decreased by one. The recursive function, *Opt*(*ru*, *g*(*i*).*startI*, *i*, *MOP*−1| {*g*(*i*).*phrase*}), is called to compute the optimal selection of the influence region of selecting *g*(*i*).*phrase* when *g*(*i*).*phrase* is selected, where *ru* is the optimal utility before *g*(*i*).*startI* (i.e., *ru*=*Opt*(0, 0, *g*(*i*).*startI*, *MOP*|{}).*ut*). Thus, the utility of the optimal selection when *g*(*i*).*phrase* is selected is *w* = *Opt*(*ru*, *g*(*i*).*startI*, *i*, *MOP*−1| {*g*(*i*).*phrase*}).*ut* + *g*(*i*).*utility*. When the utility of the

optimal selection when $g(i).phrase$ is selected is worthier than the utility without selecting $g(i).phrase$ (that is, $w > Opt(bv, si, i-1, room| CP\_List).ut$ (line 6)), the optimal selection is updated and the utility of the optimal selection is set to w. Otherwise, the optimal selection and the utility of the optimal selection are inherited from the previous results.

CP_List is not Empty

We now describe how to compute w when *CP_List* is not empty. When a phrase is selected with empty *CP_List*, *Opt* is invoked in the inference region of $g(i).phrase$. As shown in Figure 3-8, when *CP_List* is not empty, many sub-regions with different values of room have to be processed by function *Opt*. The formula of w should be designed to deal with this situation. Note that, for each phrase $g(j).phrase$ in *CP_List*, $g(j).startI$ is smaller than i. Let *phrase_n* be the number of phrases in $CP\_List \cup \{g(i).phrase\}$. Without loss of generality, the phrases in $CP\_List \cup \{g(i).phrase\}$ are sorted by their start positions in ascending order and relabeled as $\{j_1, j_2, \ldots, j_{phrase\_n}\}$, where $j_1.startI \le j_2.startI \le \ldots \le j_{phrase\_n}.startI \le i$, and $j_k.startI$ is the start index of phrase j_k. In the first sub-region from index 0 to $j_1.startI$, the maximal number of overlapping phrases allowed is *MOP*. The utility of the optimal selection of the first sub-region, which is denoted as ru_0, is $Opt(0,0,j_1.startI,MOP|\{\}).ut$. For the second sub-region from index $j_1.startI$ to $j_2.startI$, the utility of the optimal selection of the first sub-region ru_0 is taken as the base value and the maximal number of overlapping phrases allowed is $MOP-1$. Thus, $Opt(ru_0, j_1.startI, j_2.startI, MOP-1| \{j_1\})$ is called. For the third sub-region from $j_2.startI$ to $j_3.startI$ under $MOP-2$, we take the optimal utility of the previous sub-region as the base value and calculate the optimal utility in this sub-region by invoking function *Opt* in a similar manner. The above process repeats until the last sub-region from $j_{phrase\_n}.startI$ to i under $MOP-phrase\_n$ has been processed by function *Opt*. The above recurrence relation is shown as follows.

Initial condition:

$Opt(bv, si, si, room|CP\_List).ut = bv;$

Recurrence relation:

$$Opt(bv, si, i, room|CP\_List).ut$$
$$= \begin{cases} w = Opt(Opt(\dots Opt(Opt(Opt(0,0,j_1.startI, MOP|\{\}).ut, j_1.startI, j_2.startI, MOP-1|\{j_1\}).ut, \\ j_2.startI, j_3.startI, MOP-2|\{j_1,j_2\}).ut, \dots).ut, \\ j_{phrase\_n}.startI, i-1, MOP-phrase\_n|\{j_1,j_2,\dots,j_{phrase\_n}\}).ut \\ +g(i).utility \\ \text{, if } (g(i).phrase \neq null) \text{ and } (CP\_List \cup \{g(i).phrase\} \text{ is playable}) \text{ and } (room > 0) \text{ (playable), and} \\ w > Opt(bv, si, i-1, room|CP\_List).ut \text{ (worth)} \\ \\ Opt(bv, si, i-1, room|CP\_List).ut, \text{otherwise} \end{cases} \quad (7)$$

According to the above recurrence relation, we can notice that the functions with the same parameter (for example, *Opt*(0, 0, *i*, *MOP*| {}), where $1 \leq i \leq 2n-1$) are used repetitively. For saving the computation time, the result of the function with different parameters will be stored for reuse.

3.4.4 Correctness

The proposed phrase selection algorithm is designed to solve the phrase selection problem in a recursive manner by function *Opt*. We next show the correctness of the proposed phrase selection algorithm by proving the optimality guarantee of function *Opt*.

Lemma 1 Function *Opt* can always obtain the optimal selection.

We prove the correctness of function *Opt* by induction on the value of *room*.

Induction basis:

Considering *room*=1 and no playability function, the phrase selection problem is reduced to the exon-chaining or activity-selection problem. That is, no overlapping phrase is allowed and the playability function always returns true. Our algorithm is extended from the exon-chaining algorithm that the optimality has been proven in [21]. While the playability function is taken into consideration, there is no case that *g*(*i*).*phrase* is not playable with the phrases in *CP_List*. It is because that the *CP_List* is always empty when *room*=1 (that is, no overlapping phrase

exists). In addition, function *Opt* will not select *g*(*i*).*phrase* if *g*(*i*).*phrase* itself is not playable. Therefore, function *Opt* is able to obtain the optimal selection when *room*=1.

Induction hypothesis: The function *Opt* is able to obtain the optimal selection while *room* $<$ *MOP*.

Suppose *room=MOP*. In function *Opt*, the main **for** loop examines whether the new-seen phrase, *g*(*i*).*phrase*, should be selected or not. If *g*(*i*).*phrase* is not playable with the phrases in *CP_List*, function *Opt* will not select *g*(*i*).*phrase*. When *g*(*i*).*phrase* is playable with recursive-called phrase list *CP_List*, function *Opt* will recursively invoke itself on all sub-regions with smaller values of *room*. Since the value of *room* of each invocation of function *Opt* on each sub-region is smaller than *MOP*, by induction hypothesis, each invocation of function *Opt* on each sub-region is able to obtain the optimal selection. According to Equation 7, we can conclude that function *Opt* is able to obtain the optimal selection when *room=MOP*. As a result, we can prove the correctness of Lemma 1 by induction.

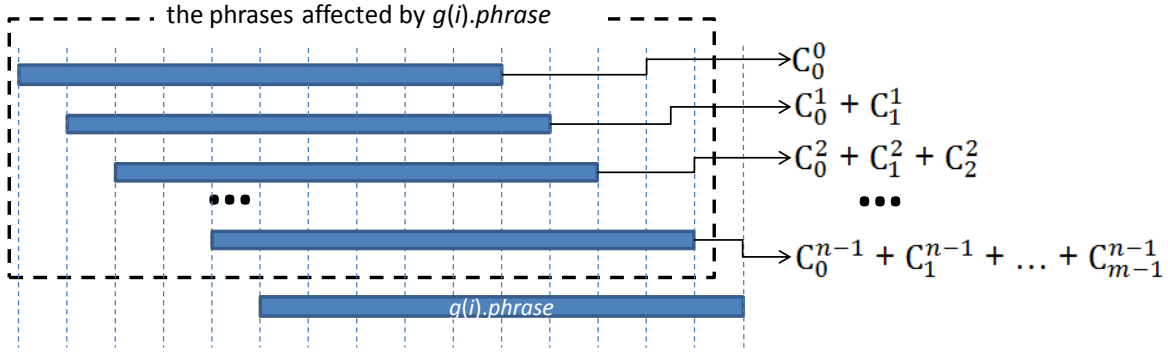


Figure 3-9. An illustration of the computation at the worst case

3.4.5 Time Complexity Analysis

The proposed phrase selection algorithm acts in a branch-and-bound manner. Each phrase is chronologically examined whether it is selected. If a phrase is selected, the optimal selection

affected by this phrase is computed. Fortunately, this process will not expand all possible changes, since the expansion process is bound at the point which the previous computation has been stored (i.e., *Opt*(0,0,*i*,*MOP*|{}), where $1 \le i \le 2n-1$). In the best case, there is no overlapping phrase and function *Opt* examines each phrase at most once. Thus, the time complexity of the proposed phrase selection algorithm is $O(\Psi \times n)$, where n is the number of phrases and Ψ is the time complexity of playability function. In the worst case, all phrases are parallel as shown in Figure 3-9. That is, each phrase overlaps with all other phrases. In the outermost invocation of function *Opt*, each phrase is checked whether it worth to be selected. For the computation of the i^{th}-seen phrase, function *Opt* recursively calls itself to examine the situation that *g*(*i*).*phrase* is selected. When no seen phrase is overlapping with the first-seen phrase, the inner *Opt* examines all possible selections and the number of possible selections is $C_0^0 = 1$. Similarly, when two phrases are seen overlapping with *g*(*i*).*phrase*, the number of all possible selections is $C_0^1 + C_1^1$ (the possible selections containing no phrase overlapping with *g*(*i*).*phrase* plus the possible selections containing one phrase overlapping with *g*(*i*).*phrase*).

Hence, the number of all possible selections when *g*(*i*).*phrase* is selected is $C_0^{i-1} + C_1^{i-1} + \ldots + C_{m-1}^{i-1} = \sum_{j=0}^{m-1} C_j^{i-1}$, where m is the maximal number of overlapping phrases allowed. Note that the number of all possible selections is bound by $m-1$ because, at most, $m-1$ phrases can be selected when *g*(*i*).*phrase* is selected. Thus, the total number of possible selections for the outermost *Opt* is $C_0^0 + (C_0^1 + C_1^1) + (C_0^2 + C_1^2 + C_2^2) + \ldots + (C_0^{n-1} + C_1^{n-1} + \ldots + C_{m-1}^{n-1}) = \Phi$. And $\Phi = \sum_{j=0}^{m-1} C_j^0 + \sum_{j=0}^{m-1} C_j^1 + \cdots + \sum_{j=0}^{m-1} C_j^{n-1} \le \sum_{j=0}^{m-1} C_j^{n-1} + \sum_{j=0}^{m-1} C_j^{n-1} + \ldots + \sum_{j=0}^{m-1} C_j^{n-1} = (n-1) \times (C_0^{n-1} + C_1^{n-1} + \ldots + C_{m-1}^{n-1}) \le (n-1) \times (C_{m-1}^{n-1} + C_{m-1}^{n-1} + \ldots + C_{m-1}^{n-1}) = (n-1) \times m \times C_{m-1}^{n-1} \le (n-1) \times m \times n^m = O(mn^m)$. Because the playability function is performed when each phrase is selected, the time complexity of the phrase selection algorithm at the worst case is $O(\Psi m n^m)$.

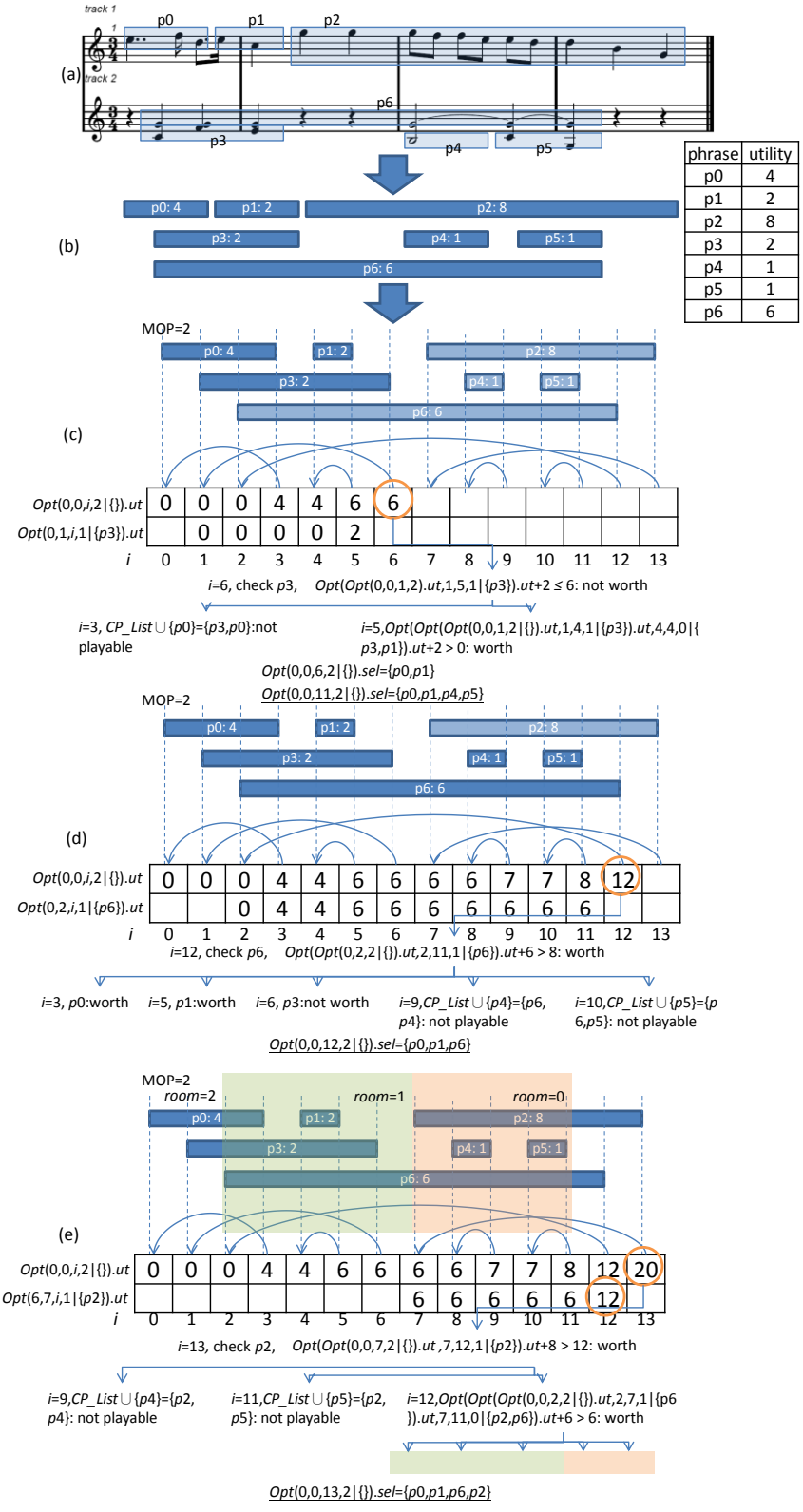


Figure 3-10. An example of the phrase selection algorithm: (a) the identified phrases in the given score; (b) the identified phrases represented by intervals; (c) a snapshot at index 6; (d) a snapshot at index 12; (e) a snapshot at index 13

Fortunately, *m* is a small number in practice. That is, the maximal number of overlapping phrases allowed by most instruments is a small constant. For example, the maximal number of overlapping phrases for a violin is 4; for most wind instruments, one; for a piano, 10; and for a guitar, six. While *m* is a small constant, the time complexity of the proposed phrase selection algorithm is polynomial time. Therefore, in practice, the execution time of the proposed phrase selection algorithm is acceptable. Interested readers can also see Table 3-9 for the execution time of the proposed algorithm on several real cases.

3.4.6 A Running Example of Phrase Selection Algorithm

Figure 3-10 shows an example of the process of the proposed phrase selection algorithm with simple playability function (the distance between the highest and lowest pitch of note cannot exceed 14 semitones) and *MOP*=2. Figure 3-10 (a) shows the identified phrases in the score; and Figure 3-10 (b) shows the phrases with utilities represented by weighted intervals. Figure 3-10 (c) depicts the result of the transformation. Since there are seven phrases, 14 indices are created. We use *Opt*(0,0,13,2|{}).*ut* and *Opt*(0,0,13,2|{}).*sel* to indicate to the optimal utility and the optimal selection, respectively. After that, three attributes of each index are extracted. For example, index 6 corresponds to the end position of phrase *p*3, and thus we have *g*(6).*phrase*=*p*3, *g*(6).*utility*=2 and *g*(6).*startI*=1. On the other hand, index 4 does not correspond to the index of the end position of any phrase, *g*(4).*phrase*, *g*(4).*utility*, and *g*(4).*startI* are null. The conditional phrase list, *CP_List*, is maintained and initialized to empty. Then, the main function *Opt*(0,0,13,2|{}) is called. The result is obtained with initial base value 0 from index 0 to 13 under the situation of at most two overlapping phrases allowed. As shown in Figure 3-10 (e), after *Opt*(0,0,13,2|{}) is finished, *Opt*(0,0,13,2|{}).*ut* = 20 and *Opt*(0,0,13,2|{}).*sel* = {*p*0, *p*1, *p*2, *p*6} are returned as the result of the proposed algorithm.

Here we use the example in Figure 3-10 to describe how function *Opt* works to compute the

optimal selection and the utility of the optimal selection. First of all, the outermost recursive function *Opt*(0,0,13,2|{}) is called to select phrases from index 0 to 13 under *room*=2 (at most two overlapping phrases are allowed). Function *Opt* goes from index 0 to index 13. The value of *g*(0).*phrase* is null since there is no phrase seen at index 0. While function *Opt* goes to index 3, it is the end position of phrase *p*0. Due to the reason that phrase *p*0 is playable and it is allowed (room $>$ 0) to select *p*0, *p*0 is selected and the function *Opt*(0,0,3,1|{*p*0}) is called to check if there is any influence of selecting *p*0. No other phrases can be seen from index 0 to 3, and thus, *p*0 can be selected (*Opt*(0,0,3,2|{}).*sel*={*p*0}) at this moment.

While determining whether phrase *p*3 (i.e., *g*(6).*phrase*) is worth to be selected, *p*3 is selected first and *Opt*(0,1,5,1|{*p*3}) is called. In solving *Opt*(0,1,5,1|{*p*3}), it meets *p*1 and *p*2 because *g*(3).*phrase* and *g*(5).*phrase* are not null. However, *p*0 (i.e., *g*(3).*phrase*) and the phrase in the *CP_List* (i.e., *p*3), are not playable due to the reason that the overlapping part of phrase *p*0 and *p*3 (C4 and E5) exceeds the *Through_hand* threshold. In contrast, *g*(5).*phrase* is playable with the phrase in *CP_List*. In addition, *g*(5).*phrase* is worth to be selected when *p*3 is selected. However, the utility of the optimal selection when *p*3 is selected is not worthier than the utility when *p*3 is not selected (i.e., *Opt*(0,1,5,1|{*p*3}).*ut*+ *g*(6).*utility*=4 $<$ *Opt*(0,0,5,2|{}).*ut*=6). Thus, *Opt*(0,0,6,2|{}).*ut* is set to 6. As shown in Figure 3-10 (c), the optimal selection *Opt*(0,0,6,2|{}).*sel* is {*p*0,*p*1}. When function *Opt* goes to index 12, the utility of the optimal selection when *p*6 is selected (i.e., *Opt*(*Opt*(0,0,2,2|{}).*ut*,2,11,1|{*p*6}).*ut*+6=12) is worthier than the utility when *p*6 is not selected (i.e., *Opt*(0,0,11,2|{}).*ut*=8). Thus, *Opt*(0,0,12,2|{}).*ut* is 12 and *Opt*(0,0,12,2|{}).*sel* is {*p*0,*p*1,*p*6}.

Consider the example that phrase *p*2 is checked in Figure 3-10 (e). Function *Opt*(6,7,12,1|{*p*2}) is called to examine the influence of the optimal selection under the condition that *p*2 is selected. During the process in *Opt*(6,7,12,1|{*p*2}), *p*4, *p*5 and *p*6 will be

examined sequentially. Phrase *p*4 and *p*5 are not playable with the phrase in *CP_List* (i.e., *p*2), while *p*6 is playable with the phrase in *CP_List* and *room*＞0 (the value of *room* is 1). Now function *Opt* examines whether *p*6 is worth to be selected by considering *p*6 with *CP_List*. At this moment, *CP_List*∪*g*(12).*phrase* contains two phrases (*p*2 and *p*6). Then, *p*2 and *p*6 are sorted and relabeled according to their start positions. Thus, j_1=*p*6 and j_2=*p*2. There are three sub-regions: the sub-region from 0 to j_1.*startI*=2 with *room*=*MOP*=2, the sub-region from j_1.*startI*=2 to j_2.*startI*=7 with *room*=1, and the sub-region from j_2.*startI*=7 to *i*－1=11 with *room*=0. The utility of the first sub-region (from 0 to 2 with *room*=2) *Opt*(0,0,2,2|{}).*ut*=0 is computed first. The optimal utility of the first sub-region is taken as the base value of function *Opt*(0,2,7,1|{*p*6}) for computing the optimal utility of the second sub-region (from 2 to 7 with *room*=1). After obtaining *Opt*(0,2,7,1|{*p*6}).*ut*=6, it is taken as the base value for the third sub-region. Similarly, we compute the utility of the third sub-region (with base value 6 from 7 to 11 with *room*=0) by calling *Opt*(6,7,11,0|{*p*6,*p*0}). Since phrase *p*6 is worth to be selected under the condition that phrase *p*2 is selected ($w > 6$), *p*2 is selected. Back to the outermost *Opt*, phrase *p*2 is also worth to be selected. Thus, the final optimal selection is {*p*0, *p*1, *p*6, *p*2} and the utility of the optimal selection is 20.

Table 3-3. Parameters for SVM

Parameter	Value
The exponent for the polyphomial kernel	1
Gamma for the RBF kernel	0.01
Sets the size of the kernel cache (a prime number)	250007
Sets the tolerance parameter	1.0e-3
Sets the epsilon for round-off error	1.0e-12
The complexity constant C	1

3.5 Experiments of Proposed Music Arrangement Framework

According to the proposed automatic music arrangement framework, we design an automatic

music arrangement system for piano in our experiments. Our music arrangement system was implemented in Java, along with two open source packages, jMusic [47] and Weka [53]. The library, jMusic, provides an environment for manipulating MIDI data; Weka provides machine learning tools for our training and test process. We choose MIDI-format music as a source of symbolic data. All the music data we collected are available on the web page (http://mpc.cs.nctu.edu.tw/~stevechiu/mas/mas_work/).

3.5.1 Effectiveness of Arrangement Element Determination

In implementing of the arrangement element determination, we chose the support vector machine (SVM) [6] as our classifier. As mentioned in Section 3.2, the modified SVM is a five-class classifier and is able to obtain the probabilities over five classes.

We collected the segmented tracks by first performing track segmentation on each music object in our database. Two musically trained experts were then asked to annotate the type of arrangement element for some of the segmented tracks. Both of them have received at least 15-year music training and participate in music productions and recordings. Besides, one graduated from department of music and majored in composition and arranging. The other has five-year experience in computer music. A total of 240 segmented tracks were annotated: 78 for *foundation*, 56 for *rhythm*, 15 for *pad*, 67 for *lead*, and 24 for *fill*. The segmented tracks and their annotated result were also shown on the web page (http://mpc.cs.nctu.edu.tw/~stevechiu/mas/mas_work/showdatabase.php). We trained our classifier with the unbalanced sizes of the class because the proportion of the types of arrangement elements in a music object is also unbalanced. The parameters of SVM are set by trial and error (The values of these parameters are listed in Table 3-3). The confusion matrix of classification result is shown in Table 3-4. The f-measures for *foundation*, *rhythm*, *pad*, *lead*, and *fill* are 0.907, 0.826, 0.72, 0.813, and 0.4 respectively.

Table 3-4. Confusion matrix for five arrangement elements with tenfold cross-validation

Arrangement Element	**Classifier As**				
	fo	rh	pa	le	fi
fo=Foundation	73	3	0	2	0
rh=Rhythm	7	45	0	3	1
pa=Pad	0	3	9	2	1
le=Lead	3	1	0	61	2
fi=Fill	0	1	1	15	7

Table 3-5. Parameters for our piano arrangement system

Parameter	**Description**	**Value**
TS.τ	A threshold for track segmentation	0.5
PI.LBDM.threshold	A threshold for LBDM	0.6
UA.AE.threshold.filter (right/left hand)	A threshold to filter the phrases whose utility is too low	0.1/0.1
UA.AE.consider[fo, rh,pa,le,fi](r/l hand)	Which arrangement elements are considered	[0,0,0,1,1]/ [1,1,1,0,0]
$UA.\alpha_1$, $UA.\alpha_2$	Proportion in utility assignment	0.7, 0.3
PS.MOP (r/l hand)	Maximal overlapping phrase allowed in phrase selection	5/5
Pla.Through_Hand (r/l hand)	In playability, semitone allowed between thumb and little finger	14/14
Pla.Thumb_Index_Gap (r/l hand)	In playability, semitone allowed between thumb and point	4/4
Pla.Other_Gap (r/l hand)	In playability, semitone allowed between the other adjacent fingers	3/3

TS: Track Segmentation, *PI*: Phrase Identification, *UA*: Utility Assignment, *PS*: Phrase Selection, *Pla*: Playability, *AE*: Arrangement Element

The class, *fill*, cannot be determined very well. The properties of *fill* are very similar to *lead*, as they have common characteristics such as pitch, duration, etc. No relevant feature can be used to discriminate them. This is reason *fill* is sometimes misclassified as *lead*. According to definition, a *fill* appears between successive phrases of *lead*. The length of the phrase of *lead* is longer in most types of music; hence, most parts of *fill* are rest note. We think the major feature that can be used to distinguish *fill* from *lead* is the ratio of silence. However, in most of cases, the musician combines *fill* with the other arrangement element (usually *rhythm*) instead of adding a specific instrument performing *fill*. As such *fill* cannot be determined well. We will keep looking for relevant features with which to improve the performance of

arrangement element determination in future work.

Table 3-6. Music for experiments

Music Title	Composer	S/H
Bluesette (S1,A1)	Toots Thielemans	S
Jordu (S2,A2,P1)	Duke Jordan	S
Green Grow the Lilacs (S3,A3,P2)	N/A	S
Symphony No.5 in C minor, Op.67 Mov.4 Allgro (S4,A4)	Beethoven	S
On Springfield Mountain	N/A	S
Lakes of Pontchartrain	N/A	S
Red River Rock	Johnny & the hurricanes	S
Some Folks Do	Stephen C. Foster	S
A Virgin Unspotted	Christmas Hymn	H
'O Sole Mio	N/A (Neapolitan song)	H
Playmate / Two Little Maids	H. W. Petrie	H
The Champion	Kristopher M Thornton	H
Lazy Mary, Will You Get Up?	N/A	H
Unfortunate Miss Bailey	N/A	H
10 Little Indians	N/A	H
You're in the Army Now	N/A	H
Some Folks Do (S5,A5)	Stephen C. Foster	S
Symphony No.25 in G minor, K.183 (S6,A6)	Mozart	S

S/H: *System or human arranges*; first 16 music are used in experiment 1; S1,…,S7 are used in experiment 3 (scoring solos); A1,…,A7 are used in experiment 3 (scoring accompaniment); P1,…,P5 are used in experiment 3 (scoring playability)

3.5.2 Turing Test-like Experiment for the Arranged Results

It is difficult to evaluate the effectiveness of our music arranging system because the evaluation of effectiveness in works of art often comes down to subjective opinion. In 2001, M. Pearce proposed a method to evaluate the computer music composition system [44]. We adopted this method in designing our experiments.

The proposed system can be considered successful if the subjects cannot distinguish between the system-arranged and the human-arranged music. There were 30 subjects in total. Twenty-two subjects were composed of graduate and undergraduate students, including four subjects with at least three-year musical training affiliated with the Department of Computer Science at National Chiao Tung University. Eight subjects were music teachers at several private music schools. The prepared dataset consisted of eight human-arranged and eight system-arranged music objects. The system-arranged music was generated by our system using the parameter setting listed in Table 3-4. Confusion matrix for five arrangement elements with tenfold cross-validation

Arrangement Element	**Classifier As**				
	fo	rh	pa	le	fi
fo=Foundation	73	3	0	2	0
rh=Rhythm	7	45	0	3	1
pa=Pad	0	3	9	2	1
le=Lead	3	1	0	61	2
fi=Fill	0	1	1	15	7

. The same setting was also adopted in the succeeding experiments. The experiment used the first 16 music objects in Table 3-6. The music objects were sorted randomly and displayed to the subjects on the web page (http://www.cs.nctu.edu.tw/~scchiu/mas/survey.html). The subjects were asked to listen to each piece and determine whether it was system- or human-arranged. The proportion of correctly identified music was calculated from the obtained result, with “Mean” being the average of the accuracy. The significance test was performed with the one-sample t-test against hypothesized value 0.5 (the expected value if subjects discriminated randomly). Simply speaking, if the mean value is close to 0.5, we can say that it is difficult to distinguish between the system- and human- arranged music.

Table 3-7. The results of discrimination test

	Mean	SD	DF	t	P-value
All subjects	0.45	0.1453	29	-1.885	0.0695
All subjects except musically trained subjects	0.444	0.15	17	-1.61	0.1258
Musically trained subjects	0.4688	0.1423	11	-0.76	0.4635

SD: *the standard deviation*; DF: *the degree of freedom*; t: *t statistic*.

The results are shown in Table 3-7. The mean values of the three groups are close to 0.5 with around 0.15 standard deviations. According to t-test, we can accept the hypothesized value 0.5 using the 0.05 level of significance; that is, it is difficult to distinguish between the system- and human-arranged music. Considering p-value, the result of all subjects is more significant than the other two separated groups because the number of all subjects is higher. The discrimination rate of the musically trained subjects (0.4688) is a little bit higher than the discrimination rate of all the subjects excluding the musically trained subjects (0.444). Such results conform to the intuition that the musically trained subjects could discriminate with higher precision. Since the discrimination rate of musically trained subjects is still close to 0.5, we believe that it is not easy to distinguish between the system- and human-arranged music even by the musically trained subjects.

Table 3-8. The results of scoring

A: The result of scoring system-arranged piano reduction

	S1	S2	S3	S4	S5	S6	S7
Mean	0.643	1	1.143	0.571	0.929	0.5	0.214
SD	0.842	0.877	0.77	0.938	0.73	0.941	0.802

B: The result of scoring system-arranged accompaniment piano part

	A1	A2	A3	A4	A5	A6	A7
Mean	0.986	0.143	0.643	0.429	0.643	0.786	0.429
SD	0.994	1.027	1.008	0.938	0.745	0.699	0.756

C: The result of scoring playability of system-arranged music

	P1	P2	P3	P4	P5
Mean	1.182	1.273	1.364	1	0.818
SD	0.874	0.786	0.924	1.247	1.401

3.5.3 Scoring the Arranged Results

To evaluate the ability of role arrangement, five music objects were chosen, each of which was arranged into a solo and accompaniment piano arrangement. The five objects were selected from the system-arranged music list in Table 3-6, and were asterisked and assigned numbers following the music title. The original and arranged versions were put on the web page so that the subjects could listen to them alternately and comparably. The subjects were asked the question “Do you think the arrangement was successful?” The question was followed by three tips: (1) Are the original and the arrangement similar?, (2) Is the arrangement elegant?, (3) Is the arrangement like piano music?” The average score of the 22 responses was 0.714. Of the experimental music set, S7 shows the highest score. The melody and counterpoint are correctly selected for piano, demonstrating characteristics of Baroque music. In contrast, S6 had the lowest score. We think some phrases were assigned inappropriate utility, so that the other important phrases could not be selected. Furthermore, some of the selected phrases with trill technique performed by violin were not suitable for the piano. This problem may be solved by considering piano performance properties in utility

assignment.

For accompaniment, a similar question was asked, “How satisfied are you with the accompaniment of duo?” The answer contains five choices: very good (+2), good (+1), average (0), bad (-1), and very bad (-2). Only 12 subjects answered the question because some of them could not tell which accompaniment was of good quality without musical background. The mean of grade was 0.58 and standard deviation, 0.881. We think that most of the music in our dataset was suitable for being an accompaniment of duo. A7, which also shows the lowest grade among seven music objects, was the only one not suited for duo. We think too many phrases of *lead* were selected as accompaniment. The failure of arrangement element determination leads to inappropriate utility assignment, and in turn, the incorrect selection of phrases.

For playability, we displayed the sheet music of the MIDI-format arranged music by general music software with slight parameter setting for presentation. Both the arranged music and its sheet music were put on the web page questionnaire so that the subjects could listen and view simultaneously, then, assign their decisions. The instruction was “Please view the sheet music and determine if it can be played on the piano.” The five answer of choices were: 1. It is playable (+2); 2. It is playable but hard (+1); 3. Neutral (0); 4. It may not be playable (-1); 5. Absolutely, it is not playable (-2). Note that this question was optional because not all participants could read sheet music. For eight responses, the mean was 1.127 and standard deviation, 1.046. This experiment shows that the arranged results are playable.

3.5.4 Case Studies

We chose two arranged results and demonstrated the sheet music of both the original and the arranged music. Due to the limitation of space, we only take two excerpts from them to discuss.

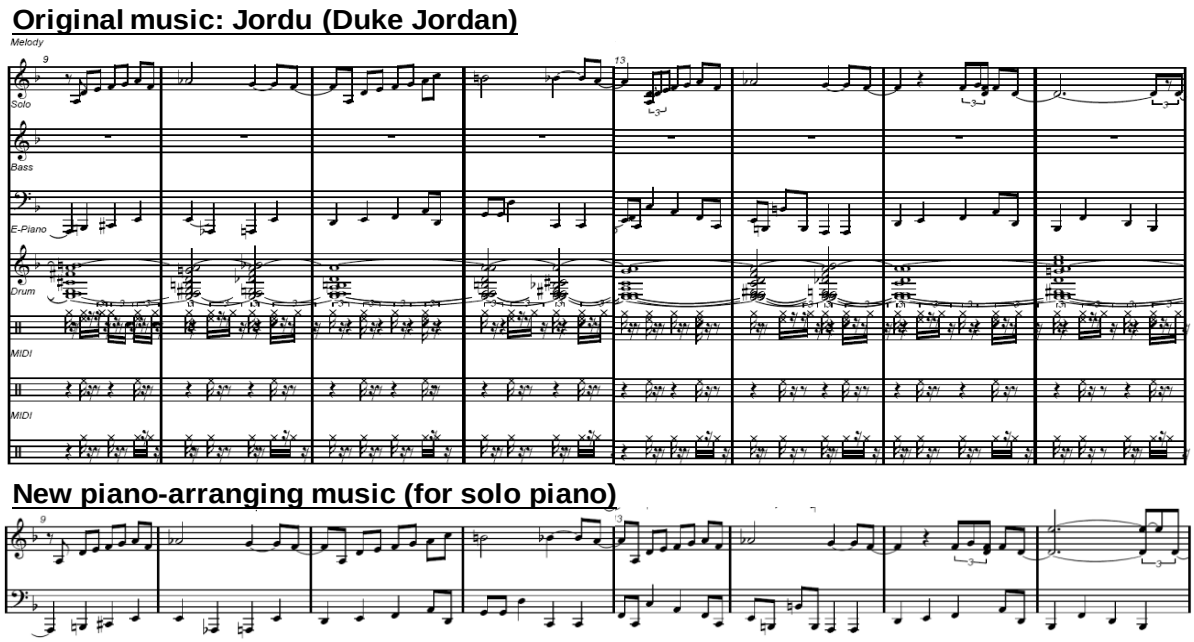


Figure 3-11. (a) Original music: excerpt from Duke Jordan "Jordu" (b) System output: piano-arranged music for a solo piano

The arranged result is jazz music, "Jordu," by Duke Jordan. The excerpts (measure 9-16) of the original and the arranged sheet music were shown in Figure 3-11a and Figure 3-11b, respectively. The ensemble comprised of four instruments: electric piano (melody and chord), vibraphone (solo), electric grand piano (chord), bass, and drum. Some instruments, such as drum, were recorded in more than one track. The system performed phrase selection for the right hand then the left hand. The system demonstrated the ability to select the correct melody for the right hand because the arrangement element determination contributed to segmented tracks, which assigned the proper utility to phrases. According to the parameter setting in phrase selection, the system maximally allowed five phrases to overlap with each other. We originally anticipated that some phrases near the melody could be selected to maximize total utility. However, such did not happen because the melody and the other overlapped phrases in the chord part could not be played simultaneously. The phrases in the chord part were especially long because there was no high strength to be cut by LBDM. It needed the notes from the other phrases that could be played with the long phrase. Thus, it was difficult to find

a non-melody phrase with melody for the right hand part. Only when the phrase was short and it is playable with the melody, it was easier to be selected. An example can be seen in measure 16. An overlapping phrase with one E4 note in the chord part was selected with the melody.

In the left hand part, we found that phrase selection chose the bass part instead of the chord part because the utility of the phrase in the bass track was much larger than that of the chord track. The other non-bass phrases were selected for the left hand for the same reason. We think this result is acceptable for a piano reduction.

Figure 3-12. (a) Original music: an excerpt from a Irish folk song "Green Grow the Lilacs" (b) System output: piano-arranged music for solo piano

The other arranged result is an Irish folk song entitled "Green Grow the Lilacs." Figure 3-12a and Figure 3-12Figure 3-12b show excerpts from measure 1 to 8 of the original and the arranged versions, respectively. The original song contained five instruments: three acoustic guitars (one for melody and two for chord), bass, and violin. As can be seen, the arranged result was not just a monophonic phrase because the other phrases were playable with the main part. The phrases of melody were included correctly for the right hand part; and the arrangement for the left hand part also contained as many phrases as possible in bass and harmonic voice. We think this song was arranged successfully for a piano reduction. It was

also playable.

3.5.5 Efficiency of the Piano Arrangement System

To evaluate the response time of the system we developed, we conducted an experiment on an IBM desktop computer with a 2.4 Ghz Intel(R) Pentium(R) quad-core processor with four gigabytes of main memory running on a Linux 2.6 operating system. We show the information in process for four excerpts of the music in Table 3-9. As can be seen, it was the *overlapping phrase rate OPR*, not the length of music and the number of identified phrases, which influenced the execution time. *OPR* is the average number of overlaps between phrases. When *OPR* is high, the time complexity of phrase selection let the execution time grow polynomially.

Table 3-9. Efficiency of music arranging system

Music	length	#st	#phr	*OPR*	Round in *PS*	Execution time
Blueseet *1	0:41	7	41	7.86	734	6.06 s
Jordu *2	3:25	8	237	7.0	1343	57.702s
Lilacs *3	1:29	6	310	11.64	9552	42.817s
Sym. No.5 *4	1:05	17	112	32.28	38918	156.663s

st: *segmented track*; phr: *phrase*; PS: *phrase selection*, OPR: *Overlapping Phrase Rate*.

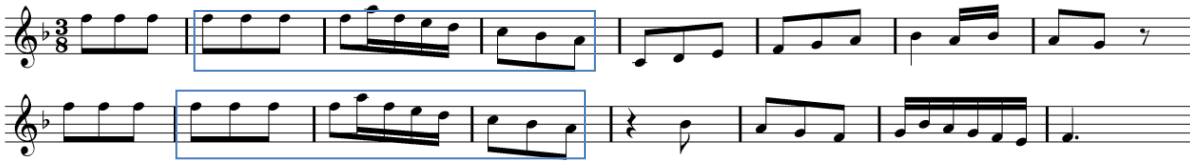
CHAPTER 4 POLYPHONIC REPEATING PATTERN MINING

4.1 Introduction

Many famous musicians ever made some descriptions or definitions for their understanding of music. For example, Edgard Varese said that music is “organized sound” and an American-born violinist, Yehudi Menuhin, mentioned that “music is art of time.” Organizing sounds over time is a design of repetition. In music theory, one of important techniques of music composition is to construct repetitive relationship among small pieces in time sequence for enhancing impression of a listener. Many researchers in musicology and music psychology fields claim that repetition is a universal characteristic in music structure modeling [48]. The segment appears repeatedly in music is so-called *repeating pattern*. As an example shown in Figure 4-1, the segment in the second block of the second line is a copy of the segment in the first block. The repeating pattern may present several meanings in music, such as *motif* and *theme*. A motif is a short musical idea which is a meaningfully recurring fragment or succession of notes. Composers usually employ the notion of motif to vary and develop whole music. In contrast to a motif, a theme is a complete phrase which is an impressive melody repeated in variation of form. In addition, depending on the composer and the type of music, it may be different for a theme in the variation extent and the repetitive frequency. Thus, the repeating pattern is an important characteristic in music.

Repeating patterns finding is useful not only for music analysis, but also for content-based music information retrieval due to both efficiency and semantic-richness requirement. That is, the repeating pattern can be used for music index, since the size of the repeating pattern is less than the size of a music object and the repeating pattern is relevant as a feature for the discrimination of music [31]. In other words, the set of repeating patterns in a music object

provides a model which is benefit for composing music according to a certain music style [9].



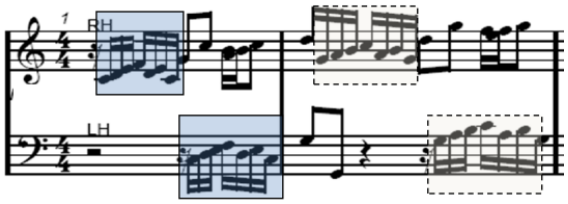
A piece of music in melody part from Mozart, Rondeau K. V. 15hh

Figure 4-1. An example of a repeating pattern

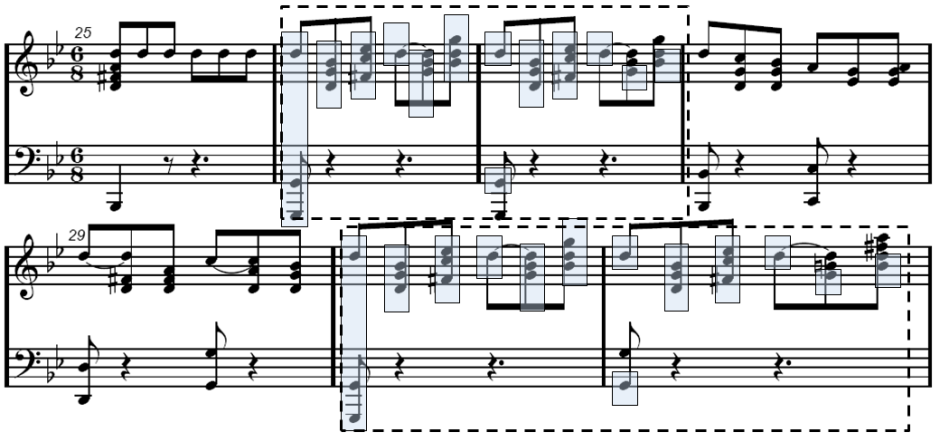
The applications of discovering the patterns occurring repeatedly first appeared in natural language field [26]. In biological field, researchers convert a DNA sequence and find the sub-string which repeats frequently in the converted string [4][46]. In multimedia area, Hsu et al. proposed in [18][18] the problem of repeating pattern mining to discover the repeating music segments. The studies of the repeating pattern mining problem firstly focused on finding exact repeating patterns in music database. However, music segments with minor difference should be regard as the instances of the same repeating pattern. Therefore, the concept of the approximate repeating pattern [18][32] and the fault-tolerant repeating pattern [25] are proposed to deal with the problem resulting from the variances among the instances of the same repeating pattern. For repeating pattern mining in music, they focus on main melody, an impressive monophonic line for listening, which can be represented by a string and propose the algorithm to find repeating patterns on a string.

The prior studies mentioned above assume that there is only one event (note) at a time. It is reasonable for a DNA sequence and music containing a clearly main melody. However, for many types of music, such as Baroque period music, etc., it may contain two or more melodies and the main melody is not clear to be found. Thereby modeling music as a string is inefficient and some significant repeating patterns may not be found. Some work addresses to this problem and develops the algorithm for mining *monophonic* patterns from *polyphonic* music. For discovering meaningfully musical patterns, they define the special type of

repeating patterns in polyphonic music, such as vertical patterns, perceptible repetitions and geometrical patterns. Conklin developed a representation of music and provides an algorithm to analyze the vertical patterns, which is benefit for representing common harmonic progresses, from one or more music objects by encoding a music object into a set of strings [13][14]. Meudic et al proposed an approach to identify the perceptible repetitions which is the similar segments located in music [40]. A geometrical pattern proposed by Meredith et al is represented in polyphonic form to find the pattern repeats in geometrical view [39].



(a) First two bars from Bach Invention No. 1 (BWV 772)



(b) A piece from Mendelssohn, song without words, Venetian Boat-Song No. 1

Figure 4-2. Two examples of polyphonic repeating patterns

To summarize, these approaches can be categorized into two scenarios: 1. discovering *monophonic* patterns from *monophonic* music; 2. discovering *monophonic* patterns from *polyphonic* music. However, there is a problem in traditional repeating pattern mining approaches. With polyphonic music from Baroque era, for instance, there may be two or more voices that play melodic line simultaneously, and the same piece often appears in different voices. As an example of piano music in Figure 4-2.a, in this case, one melody occurs in the

treble clef overlapped by another melody in the bass clef. Specially, the repeating pattern in the box appears interchangeably between two staves. Main melody extraction cannot be used in this case. Besides, it would be suitable to describe an impressive piece of music in polyphonic form, rather than in monophonic form. In Figure 4-2.b, an example of a repeating pattern in polyphonic form is shown in the successively rectangular box.

In this study, we propose the approaches to discover *polyphonic* repeating patterns in *polyphonic* music data modeled as a set-sequence data. To discover patterns from the set-sequence data, we give a formal definition on polyphonic repeating pattern discovery problem, which is also a generalized problem of traditional repeating pattern discovery. To mine polyphonic repeating patterns, we first propose a level-wise mining algorithm, named ***A-PRPD*** (standing for Apriori-based Polyphonic Repeating Pattern Discovery), based on anti-monotonic property[3]. The approach *A-PRPD* finds patterns by joining shorter frequent patterns. Since it takes too much time for *A-PRPD* to check every pair of frequent patterns whether it can be used to generate candidates or not, we propose an algorithm ***D-PRPD*** (standing for Depth-first-search based Polyphonic Repeating Pattern Discovery) to avoid this problem. In *D-PRPD*, each candidate is generated by two types of extension directly instead of pair wise check for candidate generation in *A-PRPD*.

Another issue is frequency counting for both two algorithms because they have to count frequency by sequence scan for every candidate pattern. Such phenomenon makes two algorithms spend much time in sequence scan, thereby making them not suitable for long sequence. In view of this, we develop the ***bit-string approach*** to reduce sequence for frequency counting. The positions of the occurrences of a polyphonic repeating pattern are recorded by a bit-string, called the bit-string index. Then, we design the bit-string operation to

[3] If a pattern is frequent, then all its sub-patterns are frequent.

derive the bit-string of the longer polyphonic repeating pattern and also its frequency by basic hardware operations, specifically, *SHIFT* and *AND* operations. By utilizing the bit-string approach in two algorithms, the number of sequence scan is reduced, thus speeding up the process of mining polyphonic repeating patterns. To measure the performance of *A-PRPD*, *D-PRPD* and their improvements, several experiments are conducted on both real dataset and synthetic dataset. The experimental results show that the bit-string approach improves both two algorithms and *D-PRPD* with the bit-string approach is able to discover polyphonic repeating patterns efficiently than others, showing the better scalability of *D-PRPD* with the bit-string approach over others.

4.2 Preliminary of Polyphonic Repeating Pattern

4.2.1 Problem of Polyphonic Repeating Pattern Mining

In this section, we formulate the polyphonic repeating patterns discovery problem. Both of a music object and a pattern are represented as a set-sequence. A set-sequence is a representation which collects sets of the musical notes appearing at the same time in chorological order.

Definition 4-1 Let $I=\{i_1,i_2,\ldots,i_n\}$ be a set of elements, a *set-sequence* ***sd***$=<s_1,s_2,\ldots,s_m>$ is an ordered list of set, where $s_i \subseteq I$ and $s_i \neq \varnothing$, $i \in \{1, 2, \ldots, m\}$. The ***size***, *m*, of a set-sequence is the number of set in *sd*. The ***length*** of a set-sequence is defined as $l=\sum_{1}^{m}|s_i|$, where $|s_i|$ denotes the cardinality of the set s_i. In other words, the length of a set-sequence is the total number of elements in a set-sequence.

Example 4-1 Consider a set-sequence $sd=<\{A, E\}, \{C, D, F\}, \{D\}>$, there are three sets in *sd*, so the size of *sd* is 3. Since the first set has 2 elements, the second has 3 and the third has 1; the length of *sd* is 2+3+1=6.

Definition 4-2 (k-position instance) Given two set-sequences, $sp=<p_1, p_2,\ldots, p_i>$ and $sd=<q_1, q_2,\ldots, q_j>$, where $size(sp) \leq size(sd)$, if there exists a k such that $p_1 \subseteq q_k$, $p_2 \subseteq q_{k+1}$, …, $p_i \subseteq q_{k+i-1}$, where $k+i-1 \leq j$, we call that sp has a ***k-position instance*** in sd. To discriminate these two set-sequences, sp is called a *set-sequence pattern* and sd is called a *set-sequence data*.

We can define the polyphonic repeating pattern by using the above definition.

Definition 4-3 Given a set-sequence pattern sp and a set-sequence data sd, we use ***freq***(sp,sd) to denote the **frequency** of a set-sequence; that is, the number of different *k-position instances* in sd with respect to sp. If $freq(sp,sd) \geq t$, where t is a user-defined threshold, the set-sequence sp is a ***polyphonic repeating pattern*** or ***frequent pattern*** in short.

Table 4-1. All polyphonic repeating patterns discovered from the set-sequence data <{A,E}, {C,D,F}, {D}, {B}, {A,B}, {A,B,C,F}, {C,D}>. (PRP: Polyphonic Repeating Pattern)

PRP	*freq*	PRP	*freq*
<{A}>	3	<{C, C}>	2
<{B}>	3	<{C}, {D}>	2
<{C}>	3	<{F}, {D}>	2
<{D}>	3	<{A}, {C, D}>	2
<{A, B}>	2	<{A}, {C, F}>	2
<{A}, {C}>	3	<{A}, {F}, {D}>	2
<{A}, {F}>	2	<{C, F}, {D}>	2
<{B}, {B}>	2	<{A}, {C, F}, {D}>	2
<{C, F}>	2		

Example 4-2 Consider an example set-sequence sd = <{*A*, *E*}, {*C*, *D*, *F*}, {*D*}, {*B*}, {*A*, *B*}, {*A*, *B*, *C*, *F*}, {*C*, *D*}> and let threshold t be 2. All polyphonic repeating patterns of sd are shown in Table 4-1. There are 15 polyphonic repeating patterns in this set-sequence data. If we set t to 3, the discovered polyphonic repeating patterns are <{*A*}>, <{*B*}>, <{*C*}>, <{*D*}> and <{*A*}, {*C*}>.

Finally, we define the polyphonic repeating pattern discovery problem as follows.

Problem Statement Given a set-sequence data *sd* and a user-specified threshold *t*, the task is to find all polyphonic repeating patterns in *sd*.

The set-sequence representation is an extension of a string which is used to represent a music object in traditional repeating patterns mining. When a music object is represented in a set-sequence, we can capture harmonic and counterpoint information which the string representation cannot do. Note that compare to sequential pattern mining problem [3], the problem of mining polyphonic repeating patterns treats a set-sequence data as a database and defines a different pattern in a set-sequence. When applying sequential pattern mining algorithm to find the polyphonic repeating patterns, the music object has to be divided into a set of set-sequence. However, to divide a music object is unreasonable because polyphonic repeating patterns may appear anywhere. Hence, sequential pattern mining algorithm can not apply to this issue.

4.2.2 Music Representation

In this chapter, a *music object* is represented by a sequence of sets, i.e., ***set-sequence***. Preprocessing of music data is composed of two steps. First, the quantization process is performed. This process is to adjust the onset time and note duration of each note to reasonable rhythmic unit. An example is given and shown in Figure 4-3. This process is necessary. Because the time resolution of note in symbolic music data is usually high for flexibility of expression, some notes may not appear in rhythmic position precisely. Second, the notes occurring in the same time are grouped and the sets are ordered in time sequence. After preprocessing, a set-sequence of notes is generated.

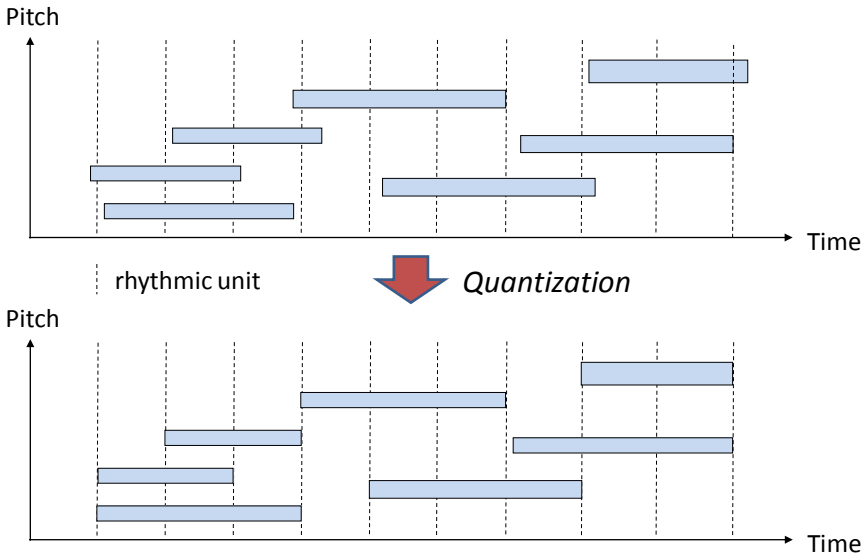
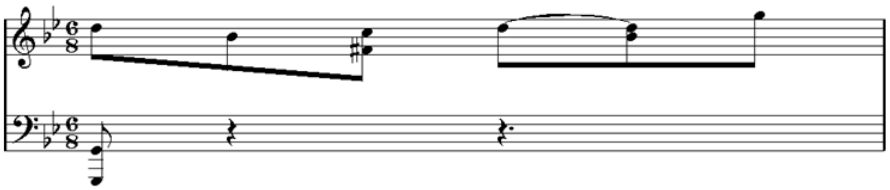


Figure 4-3. An example of quantization process

To discover musical patterns, we consider two attributes of a note. Two of them are *pitch* and *duration*. We use $sd_{(pitch,\ duration)}$ to denote a set-sequence data represented in 2-tuple. In *pitch*, two types of value are used, *exact pitch value* (EPV) and *pitch interval* (PI). For EPV, a note is recorded by its exact pitch value. For PI, all intervals between successive two sets are recorded. For attribute *duration*, *exact duration value* (EDV) is employed; that is, the number of beats of a note sustains. We give an example of variant representations in Figure 4-4. We represent exact pitch value following MIDI format, i.e., the MIDI number of center C is 60, etc. On the other hand, the extract duration value of the note is 0.5 when the note is an eighth note. The elements in first set under (EPV, EDV) representation are (31, 0.5), (43, 0.5) and (74, 0.5). For (PI, -) representation, the elements in first set are the interval between the first set and the second set, i.e., (49, -), (37, -) and (-4, -). For simplification, we use an alphabetic symbol to denote identical 2-tuple elements (*pitch*, *duration*) while describing the problem and the proposed algorithms.



Note name	…	G0	…	G1	…	C3	C#3	D3	…	C4	C#3	…
MIDI number	…	31	…	43	…	60	61	62	…	72	73	…

(EPV, EDV)	< {(31,0.5), (43,0.5), (74,0.5)}, {(70,0.5)}, {(66,0.5), (72,0.5)}, {(74,1)}, {(70,0.5)}, {(79,0.5)} >
(PI,-)	< {(39,-), (27,-), (-4,-)}, {(-4,-), (2,-)}, {(8,-), (2,0)}, {(-4,0)}, {(9,-)} >

Figure 4-4. An example of variant representations

A common repeating pattern can be found by considering *exact pitch value* and *duration*. By employing these representations in this study, some kinds of motif development defined by music theorem [49] can be found by discovering polyphonic repeating pattern in a music object represented in some combinations of these features. For instance, one of the most important motif developments, *Sequence*, can be found by considering *pitch interval* and *duration*. One point needed to mention is that there are two kinds of transposition, real transposition and tonal transposition. The main difference of these two transpositions is made by naturally occurring half steps (abbreviated by *NOHS*) in musical scale. For example, in C major scale the pitch interval between C-D and E-F are different in tonal transposition, but they are viewed as the same distance in real transposition. Real transposition is not affected by *NOHS*, it keeps the intervallic structure exactly. On the other hand, the quality of the interval structure in tonal transposition is also flexible to fit *NOHS* in musical scale. Therefore, we consider both two types of representations in pitch interval.

Algorithm A-PRPD(*sd*,*t*)
Input: *sd*, *t*
Output: polyphonic repeating pattern set

1: L_1={sp_1 |all frequent length-1 patterns};
2: **for** (k=2; $L_{k-1} \neq \phi$; k++)
3: C_k=*pattern-extend*(L_{k-1});
4: **foreach** length-*k* candidate pattern $c \in C_k$ **do**
5: *freq-count*(*c*, *sd*);
6: L_k={ $c \in C_k \mid freq(c, sd) \geq t$ }
7: output all frequent patterns;

freq-count(*sp*,*sd*)
Input: a set-sequence pattern *sp*, a set-sequence data *sd*
Output: frequency

1: *freq*=0;
2: **for**(*i*=1; $i \leq (sd.size - sp.size + 1)$; *i*++) **do**
3: **if** *sp* has an *i*-position instance in *sd* **do**
4: *freq*++;
5: **return** *freq*;

pattern-extend((*k* −1)-*RPRSet*)
Input: a set of length-(*k* −1) patterns
Output: a set of length-*k* candidate patterns

1: **for** all pair (*sp*1, *sp*2) in (*k* −1)-*RPRSet* **do**
2: *sp*1'= *sp*1 with no first element;
3: *sp*2'= *sp*2 with no last element;
4: **if** *sp*1' is equal to *sp*2' **then**
5: **if** the cardinality of the last set of *sp*2 is 1 **then**
6: *c_sp* =append the last set of *sp*2 to *sp*1;
7: **else then**
8: *c_sp* =add the last element of *sp*2 to *sp*1's last set
9: **if** *c_sp.length*=*k* **then**
10: add *c_sp* to C_k
11: **return** C_k;

Figure 4-5. *A-PRPD* algorithm

The set-sequence representation is a type of a piano-roll or a string-based representation for music. This representation is suitable for homophonic music[4] [30]. As to polyphonic music, it can be represented sufficiently by combining the notes with two attributes, pitch and duration, occurring at a short period of time into a set. Comparing to the geometric representation [39],

[4] Comparing to polyphonic music, in monophonic music, all parts move in parallel rhythm and pitch.

the set-sequence representation is more rigid showing less flexibility in discovered patterns. But, without missing some significant patterns, the set-sequence representation can focus on the patterns occurring repeatedly over time domain to avoid plenty of irrelevant patterns occurring in geometric view.

4.3 Mining Polyphonic Repeating Patterns

Two approaches, *A-PRPD* and *D-PRPD*, are proposed for mining polyphonic repeating patterns from a set-sequence data.

4.3.1 Apriori-based Polyphonic Repeating Pattern Discovery (*A-PRPD*)

Algorithm *A-PRPD* (Apriori-based Polyphonic Repeating Pattern Discovery) is a level-wise approach based on Apriori property [2] to discover polyphonic repeating patterns from music data. The patterns are generated step by step from short length pattern to long length one. The frequent patterns with short length are discovered first and used to generate longer patterns. For the description of the process, we denote the set of length-k polyphonic repeating pattern by C_k. The frequent patterns are collected from C_k is denoted by L_k. The main process makes multiple passes over data. The k-th pass generates length-k patterns. The first pass, find L_1 by checking if the frequency of each possible length-1 pattern is larger or equal to the threshold t. The subsequent passes consist of two steps. In the k-th pass, the first step generates the set of length-k candidate patterns C_k from L_{k-1} by employing *pattern-extend* method described later. Then, the second step checks frequency *freq* for each candidate pattern in C_k and finds the set of length-k frequent patterns L_k. The subsequent pass repeats until L_k is empty. The answer is the union of all frequent patterns in each pass. The detailed algorithm is shown in Figure 4-5. The *pattern-extend* is used to generate all possible patterns C_k from L_{k-1}. We borrow the

concept from anti-monotonic principle[5]. For example, if a length-4 set-sequence pattern <{*A*}, {*A*,*D*,*E*}> is frequent, then the length-*3* set-sequence patterns, <{*A*}, {*A*,*D*}> and <{*A*,*D*,*E*}>, must be frequent. Thus, we check all pairs in L_{k-1} to check if there are length-*k* patterns.

Hence, *pattern-extend* procedure is designed as follows. Let *sp*1=<{$e_{1,1}$, $e_{1,2}$, …, $e_{1,n1}$}, {$e_{2,1}$, $e_{2,2}$, …, $e_{2,n2}$}, …, {$e_{k-1,1}$, $e_{k-1,1}$, …, $e_{k-1,nk-1}$}> and *sp*2=<{$f_{1,1}$, $f_{1,2}$, …, $f_{1,m1}$}, {$f_{2,1}$, $f_{2,1}$, …, $f_{2,m2}$}, …, {$f_{k-1,1}$, $f_{k-1,1}$, …, $f_{k-1,mk-1}$}>, then *sp*1'= *sp*1 with no the first element, i.e., <{$e_{1,2}$, $e_{1,3}$, …, $e_{1,n1}$}, {$e_{2,1}$, $e_{2,2}$, …, $e_{2,n2}$}, …, {$e_{k-1,1}$, $e_{k-1,1}$, …, $e_{k-1,nk-1}$}>; and *sp*2'= *sp*2 with no last element, i.e., <{$f_{1,1}$, $f_{1,2}$, …, $f_{1,m1}$}, {$f_{2,1}$, $f_{2,1}$, …, $f_{2,m2}$}, …, {$f_{k-1,1}$, $f_{k-1,1}$, …, $f_{k-1,mk-2}$}. For each pair of patterns (*sp*1, *sp*2) in L_{k-1}, it is checked if *sp*1' is equal to *sp*2'. If established, it means that this pair can be used to generate a length-*k* candidate pattern, *c_sp*, by adding $f_{k-1,mk-1}$ into the first set of sp_i , i.e., *c_sp* = <{$e_{1,1}$, $e_{1,2}$, …, $e_{1,n1}$}, {$e_{2,1}$, $e_{2,2}$, …, $e_{2,n2}$}, …, {$e_{k-1,1}$, $e_{k-1,1}$, …, $e_{k-1,nk-1}$, $f_{k-1,mk-1}$}>. Note that only when *c_sp.length* = *k*, *c_sp* is added into C_k, i.e., ∀e ∈ {$e_{k-1,1}$, $e_{k-1,1}$, …, $e_{k-1,nk-1}$}, $f_{k-1,mk-1} \neq$ e.

We give an example of *pattern-extend* as follows. While checking the pair of patterns in L_4, (*sp*1, *sp*2)=(<{*A*},{*C*,*D*},{*E*}>, <{*C*,*D*},{*E*,*F*}>), we compute *sp*1' and *sp*2', respectively. Since *sp*1' is <{*C*,*D*}, {*E*}> by deleting the first element of *sp*1 and *sp*2' is <{*C*,*D*}, {*E*}> by deleting the last element of *sp*2; *sp*1' is equal to *sp*2'. Therefore, the length-5 pattern <{*A*}, {*C*,*D*}, {*E*,*F*}> is generated from this pair of length-5 by adding the last element of *sp*2 to the last set of *sp*1 since the cardinality of the last set of *sp*2 is not 1. On the other hand, consider this pair (<{*A*,*C*,*D*}>, <{*C*,*D*}, {*E*}>) which can be used to generate a candidate after checking that these two conditions are established, the length-4 pattern <{*A*,*C*,*D*}, {*E*}> is generated from this pair by appending the last set of *sp*2 to the last of *sp*1 because of the

[5] If a frequent length-*l* set-sequence pattern, then all *l*−1 set-sequence patterns are frequent in this set-sequence pattern.

cardinality of the last set of *sp*2 is 1. By this approach of generating candidates, any possible solution would not be lost.

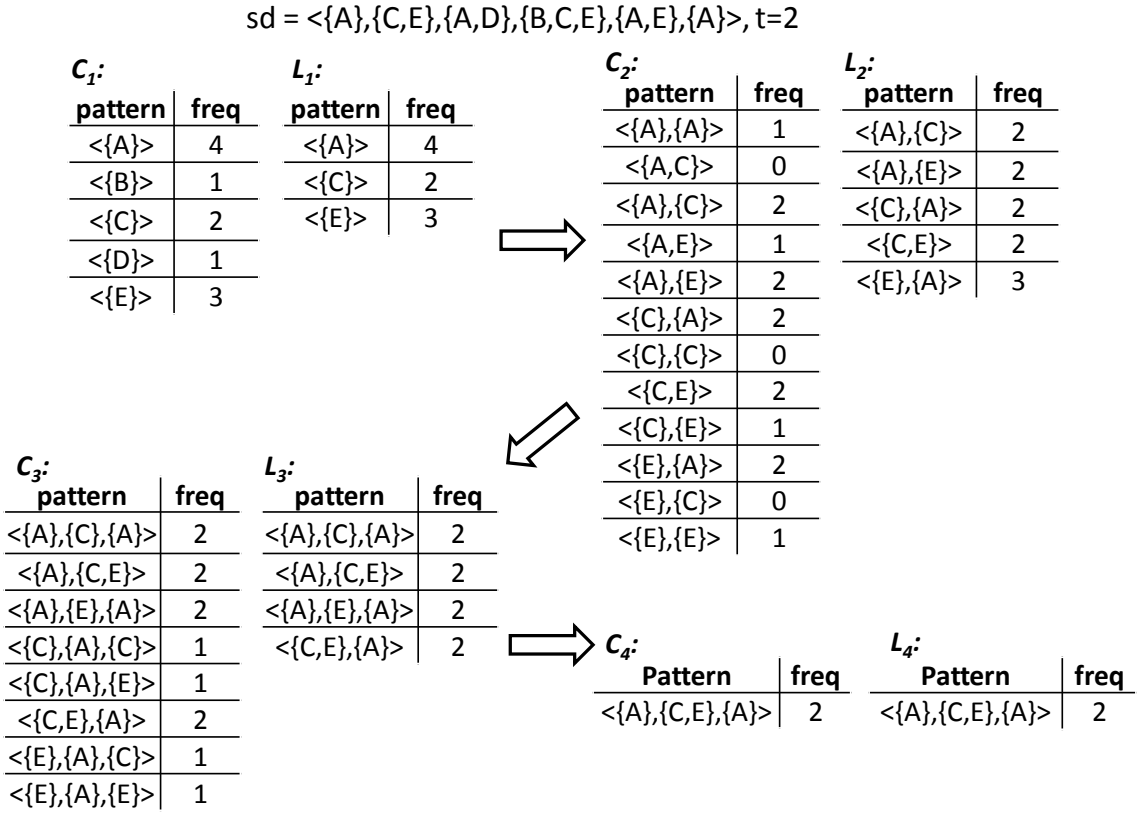


Figure 4-6. An example of running A-PRPD algorithm

An example of running *A-PRPD* is given in Figure 4-6.. A set-sequence data, <{*A*}, {*C,E*}, {*A,D*}, {*B,C,E*}, {*A,E*}, {*A*}>, is given and we assume threshold *t* is 2. L_1 is obtained by scanning the given data sequence and checking frequency for each item. The *pattern-extend* is called to generate C_2 by inputting L_1. In the process of *pattern-extend*, each pair of the patterns in L_1 will be checked, including itself, that is, *(*<{*A*}>, <{*A*}>*)*, *(*<{*A*}>, <{*C*}>*)*, *(*<{*A*}>, <{*E*}>*)*, *(*<{*C*}>, <{*A*}>*)*, *(*<{*C*}>, <{*C*}>*)* and all other pairs. There are 12 set-sequence patterns in C_2. After checking each pattern in C_2 whether it is larger or equal to 2, we will have L_2. By repeating this process, step 2 to 6 in algorithm, L_3 is derived, and then L_4. Since the empty set is generated by applying the L_4 to *pattern-extend* function, the process will be terminated. All set-sequence patterns in L_i, where i={ 1, 2, 3, 4 }, are frequent polyphonic

repeating patterns.

Algorithm T-PRPD(*sd*,*t*)
Input: *sd*, *t*
Ouput: polyphonic repeating pattern set

1: Generate root node *r_node*;
2: L_1={sp_1 |all frequent length-1 patterns};
3: **foreach** pattern sp_1 in L_1 **do**
4: Generate child *c_node* linked by *r_node* and record *sp*;
5: **foreach** child *c_node* in *r_node* **do**
6: DFS-tree(*c_node*, *ds*, L_1);
7: extract_pattern(*r_node*);

Algorithm DFS-tree(*p_node*, *ds*, L_1)

1: **foreach** pattern $sp_1 \in L_1$ **do**
2: *sp_set*=set-extend(*p_node*, *sp*);
3: *sp_seq*=sequence-extend(*p_node*, *sp*); //append the *sp* to the *p_node*
4: **if** (*sp_set.length*=*p_node.length*+1) AND (*freq*(*sp_set*,*ds*)≧t) **do**
5: Generate a node c_node_{set} linked by *p_node* and recording *sp_set*;
6: DFS-tree(c_node_{set});
7: **if** (*freq*(*sp_seq*,*ds*)≧t) **do**
8: Generate a node c_node_{seq} linked by *p_node* and recording *sp_seq*;
9: DFS-tree(c_node_{seq});
10: **return** null;

Figure 4-7. *D-PRPD* algorithm

4.3.2 DFS-based Polyphonic Repeating Pattern Discovery (*D-PRPD*)

Since observing that *A-PRPD* takes too much time to generate candidates for discovering polyphonic repeating patterns, an efficient algorithm called ***D-PRPD*** (DFS-based Polyphonic Repeating Pattern Discovery), is proposed to overcome the issue of candidate generation. A lexicographic tree is used in *D-PRPD* to provide a path to search polyphonic repeating patterns from shorter length pattern to longer one in depth-first-search manner. The frequent lenth-1 patterns are used to extend the length of the discovered frequent pattern for finding longer patterns. When In the lexicographic tree of *D-PRPD*, the node contains three types of data: *root node*, *set-sequence pattern* and *frequency*. Moreover, the height of a node means the length of the pattern in this node.

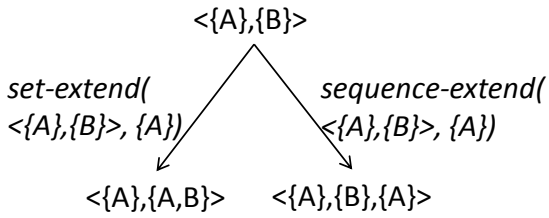


Figure 4-8. An example of two operations for pattern extension in *D-PRPD*, *set-extend* and *sequence-extend*

Algorithm description of *D-PRPD* is given in Figure 4-7. First, *D-PRPD* generates the root node with an empty set-sequence pattern and discovers all length-1 set-sequence. For each length-1 set-sequence, *D-PRPD* generates a node which stores the pattern and is linked by root node. After that, *D-PRPD* performs the *DFS-tree* to grow the node in the tree recursively. Note that *DFS-tree* method adapts depth-first-search approach to find set-sequence patterns, i.e., it will find pattern as longer length as possible until the frequency of the pattern is less than user-defined threshold *t*.

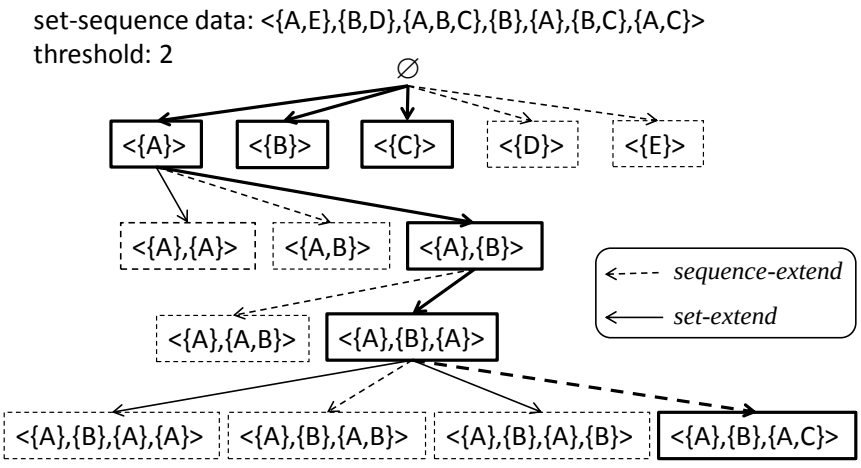


Figure 4-9. An example of a long pattern found by running *D-PRPD*

While a node generates its child node, *D-PRPD* extends the length of set-sequence by using *set-extend* and *sequence-extend* to avoid neglecting possible candidates. The set-extend operation is to extend the pattern by adding an item to the last set of a pattern. Another

operation, sequence-extend is used to extend the pattern by appending a set formed by an item to the last of the pattern. An example of these two operations is shown in Figure 4-8. However, in some circumstances, when the set-extend operation is applied, the item we added has already in the last set of the pattern. In this case, the extended pattern will be ignored because the length of the pattern does not increase. For example, as set-extend operation is performed over this pair (<{*A*,*B*},{*C*,*D*}>,{*C*}*)*, the result <{*A*,*B*}, {*C*,*D*}> is ignored.

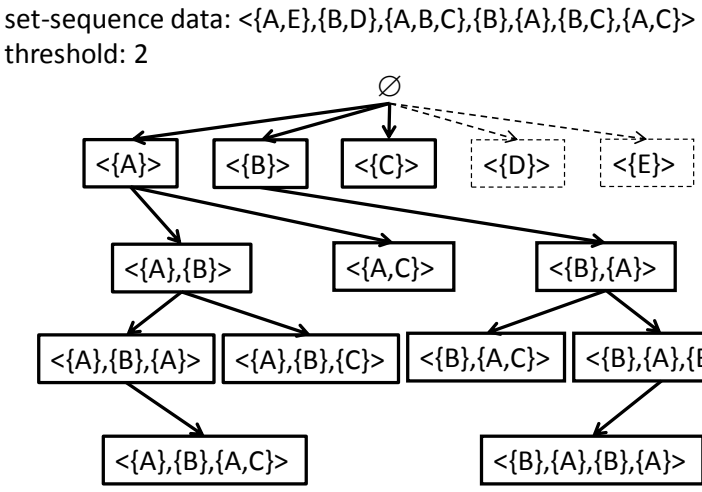


Figure 4-10. An example of all patterns found by *D-PRPD*

While a pattern is not frequent, the pattern generated in the subtree of this node will not be frequent, according to anti-monotonic property. As a result, *DFS-tree* does not grow at this node to find the longer length pattern. Instead, this procedure finds the shorter length pattern, but in different prefix. Finally, all frequent patterns will be found in lexicographic order. We give an example in Figure 4-9. Assume there is a set-sequence data *sd*=<{*A*,*E*}, {*B*,*D*}, {*A*,*B*,*C*}, {*B*}, {*A*}, {*B*,*C*}, {*A*,*C*}> and *threshold* is 2. To discover all frequent set-sequence patterns in *sd*, we first find all length-1 patterns, running as breadth first search in the lexicographic tree, and the length-1 patterns <{*A*}>, <{*B*}> and <{*C*}> are found. The frequent patterns are framed in bold line and the infrequent ones are framed in a dotted line. Then, the node of the pattern <{*A*}> is grown by extended <{*A*}> with set-extend and sequence-extend operations. But, only <{*A*}, {*A*}> is formed. After checking this pattern is

infrequent, the pattern is also eliminated. By repeating this process, we find pattern <{*A*}, {*B*}> is frequent. From this node, the pattern is extended. We find frequent pattern <{*A*}, {*B*}, {*A*}> and then <{*A*}, {*B*}, {*A,C*}> in the next level. The final result is given in Figure 4-10.

4.4 Bit-String Approach

Frequency counting is the main performance issue of our proposed algorithms. To improve the performance, a bit-string approach is developed for counting frequency efficiently.

	sd=<{A,E},	{B,D},	{A,B,C},	{B},	{A},	{B,C},	{A,C}>
<{A}>	1	0	1	0	1	0	1
<{B}>	0	1	1	1	0	1	0
<{C}>	0	0	1	0	0	1	1
<{A,C}>	0	0	1	0	0	0	1
<{A},{B},{A}>	1	0	1	0	1	0	0

Figure 4-11. An example of bit-string index

4.4.1 Bit-String Index

A set-sequence pattern *sp* in a set-sequence data *sd* can be represented by a bit-string. The length of the bit-string *bs*(*sp*) is equal to the size of *sd*. The *k-th* value of *bs*(*sp*) is 1 when *sp* has a *k-position instance* in *sd* ; otherwise, *k-th* value is 0. Furthermore, the total number of bit "1" in the bit string is equal to the frequency of this pattern. We give the formal definition of the bit-string index as follows.

Definition 4 (*bit-string index*) Given a set-sequence pattern $sp=< p_1, p_2, \ldots, p_i >$ and a set-sequence data $sd=< q_1, q_2, \ldots, q_j >$, where $size(sp) \leqq size(sd)$ and $size(sd)$ is j. We say the bit-string of *sp* in *sd* is $bs(sp)= b_1\ b_2\ \ldots\ b_j$, where b_k=1, if b has a *k-position instance* in *sd*; otherwise, b_k=0.

An example of bit-string index is given in Figure 4-11. For a sequence pattern *sp*=< {*A*}, {*B*}, {*A*} >, it has k-position instances in *sd* at 1, 3 and 5; thus, *bs*(< {*A*}, {*B*}, {*A*}>)=1010100.

4.4.2 Frequency Counting with Bit-String Operation

While the bit strings are maintained, the frequency of the extended pattern can be counted efficiently by applying bit-string operation. We denote the bit string of the length-*l* pattern as $bs(sp_l)$.

Bit-String Approach used in A-PRPD

In the proposed *A-PRPD* algorithm, *pattern-extend* function is used for finding longer polyphonic repeating patterns. That is, a length-*l* pattern *c_sp* is generated by two length-(*l*-1) pattern $sp1_{l-1}$ and $sp2_{l-1}$. As previous mentioned, we have already maintained their bit strings, $bs(sp1_{l-1})$ and $bs(sp2_{l-1})$. The *pattern-extend* function is performed when $sp1'_{l-1}$ ($sp1_{l-1}$ deleting the first element) and $sp2'_{l-1}$ ($sp2_{l-1}$ deleting the last element) are equal. Depended on the cardinality of first set of $sp1_{l-1}$ (larger than 1 or equal to 1), the size of $sp1'_{l-1}$ and $sp2'_{l-1}$ are equal to $sp1_{l-1}$.size or $sp1_{l-1}$.size－1. When the cardinality of the first set of $sp1_{l-1}$ is larger than 1, then the first set of $sp2_{l-1}$ is included by the first set of $sp1_{l-1}$; otherwise, the first set of $sp2_{l-1}$ is included by the second set of $sp1_{l-1}$. According to *pattern-extend* operation, c_sp_l is equal to the pattern which the last element of $sp2_{l-1}$ is added to the last set of $sp1_{l-1}$ or is appended to $sp1_{l-1}$. We discuss these two cases for designing bit-string operation as follows.

Case 1: Since the second set of $sp2_{l-1}$ is comprised by the first set of $sp1_{l-1}$, a property is in the extended pattern sp_l: the element *i* used to extend will occur at the *size*'-th position. We know $bs(sp2_{l-1})$ records all *k*-positions instances in *sd*. It also denotes that, for each *k*-position instances, there is an element *i* appearing after *size*' or *size*'－1 position. Thus, $bs(sp_l)$ can be derived by checking all k-position instances of $sp1_{l-1}$ whether there is an element appearing

after *size*' or *size*'－1 position. The examination can be accomplished by using two hardware operations, bitwise-and-operation (denoted by '&') and bitwise-shift-operation (denote by *LEFT_SHIFT*), as the formula: $bs(sp1_{l-1})$ & $LEFT\_SHIFT_1(bs(sp2_{l-1}))$.

Case 2: Since the first set of $sp2_{l-1}$ is included by the first set of $sp1_{l-1}$, it not necessary to shift $bs(sp2_{l-1})$ to align $bs(sp1_{l-1})$. Thus, $bs(sp_l)$ can be derived by $bs(sp1_{l-1})$ & $bs(sp2_{l-1})$.

To summarize, the bit-string of length-*l* pattern, $bs(sp_l)$ can be derived by the following formula.

$$bs(sp_l) = bs(sp1_{l-1}) \ \& \ LEFT\_SHIFT_i(bs(sp2_{l-1})) \quad (8)$$

where i=0, if the cardinality of the first set of $sp1_{l-1}$ is equal to 1; otherwise, i=0.

For example, given $sp1_3$ = <{*A*}, {*B*}, {*A*}>, $sp2_3$ = <{*B*}, {*A*,*C*}>, and their bit-string representations are $bs(sp1_3)$ = 1010100, $bs(sp2_3)$ = 0100010. After checking, these two length-3 patterns can be used to generate a candidate sp_4 = <{*A*}, {*B*}, {*A*,*C*}>. We need to check the frequency of sp_4. The process of calculating the frequency of this sp_4 by bit-string operation is described as follows. First, we obtain $LEFT\_SHIFT_1(bs(sp2_3))$ = 1000100. After that, we perform AND operation with $bs(sp1_3)$ = 1010100 and $LEFT\_SHIFT_1(bs(sp2_3))$ = 1000100. Hence, we obtain $bs(<\{A\}, \{B\}, \{A,C\}>)$=1000100. Consequently, the frequency of sp_4 is 2 by counting '1' of the bit-string.

Bit-String Approach in D-PRPD

In *D-PRPD*, a candidate pattern is generated from the pattern in parent node and length-1 pattern by set-extend or sequence-extend operation. For descriptions, we discuss *sequence-extend* operation first, then, *set-extend* operation.

(a) *sequence-extend* operation

The sequence-extend operation extends a length-(*l*-1) pattern by appending a size-1 pattern.

According to definition, we know sp_{l-1}=< s_1,…, s_{m-1}> and sp_l=< s_1,…, s_{m-1}, {i}>, where s_i is a set and $\sum_{1}^{m-1}|s_i| = l-1$. The *sequence-extend* is to attached the size-1 set to the pattern; that is, sp_l has one more set {i} than sp_{l-1}, which appears at m-th position. If we know which k-position instances of sp_{l-1} in sd appearing {i} after m position(s), then, k-position instances of sp_l is derived. As we known, $bs(sp_{l-1})$ and $bs(sp_1)$ record all position instances of sp_{l-1} and sp_1, respectively. The behavior "checking every k-position instances of $bs(sp_{l-1})$ whether has sp_1 (i.e. {i}) after m position(s)" is equal to Equation (9).

$$bs(sp'_l) = bs(sp_{l-1}) \,\&\, LEFT\_SHIFT_i(bs(sp_1)) \tag{9}$$

where i is the position of the last set of sp_{l-1}.

For example, given sp_{l-1}=<{A}, {B}, {A}>, sp_1=<{B}>, $bs(sp_{l-1})$=1010100, $bs($<{B}>$)$=0111010, and we apply *set-extend*(<{A}, {B}, {A}>, <{B}>) to derive $bs($<{A}, {B}, {A}, {B}>$)$. In this case, the position of the last set of sp is 3, so $bs($<{B}>$)$ has to left shift 3 positions, and we obtain $LEFT\_SHIFT_2(bs(sp_1))$=1010000. Then, performing AND operation between $bs(sp_{l-1})$=1010100 and $LEFT\_SHIFT_2(bs(sp_1))$=1010000 will derive $bs($<{A}, {B}, {A}, {B}>$)$=1010000. After aggregating '1' in the bit string, frequency of sp'_l is 2.

(b) *set-extend* operation

Since sp_{l-1}=< s_1,…, s_{m-1}> and sp_1=< {i} >, by set-extend(sp_{l-1}, sp_1) , we have sp_l=< s_1,…, s_{m-1} ∪ {i}>. In sp_l, the position of the last set which {i} is included is m-1 position after the first set. The set {i} is less one position than {i} in *sequence-extend*. Thus, with the same reason, we check every k-position instances of $bs(sp_{l-1})$ if sp_1 (i.e. {i}) appears m position(s) later. In *set-extend* operation, compare to *sequence-extend* operation, sp_1 is added to the last set of sp_{l-1}, consequently, sp_1 needs to be shifted left less 1 position than the position of sp_1 in *set-sequence* operation be shifted.

In set-extend operation, the formula is designed as Equation (10).

$$bs(sp'_l) = bs(sp_{l-1}) \ \& \ LEFT\_SHIFT_{i-1}(bs(sp_1)) \quad (10)$$

where i is the position of the last set of sp_{l-1}.

For example, given sp=<{A}, {B}, {A}>, sp_1=<{C}>, and their bit string are $bs(sp_{l-1})$=1000100, bs(<{C}>)=0010011. We apply *set-extend*(<{A}, {B}, {A}>, <{C}>) to derive bs(<{A}, {B}, {A,C}>). In this case, the position of the last set of sp is 3, so bs(<{C}>) has to left shift 3−1 positions, and we obtain $LEFT\_SHIFT_2(bs(sp_1))$=1001100. Then, performing AND operation between $bs(sp_{l-1})$=1000100 and $LEFT\_SHIFT_2(bs(sp_1))$=1001100 will derive bs(<{A}, {B}, {A,C}>)=1000100. After aggregating '1' in the bit string, the frequency of sp' is 2.

4.5 Experiments of Polyphonic Repeating Pattern Mining

To show efficiency of our approaches, a series of experiments are conducted. We also show the effectiveness of our approaches.

4.5.1 Efficiency

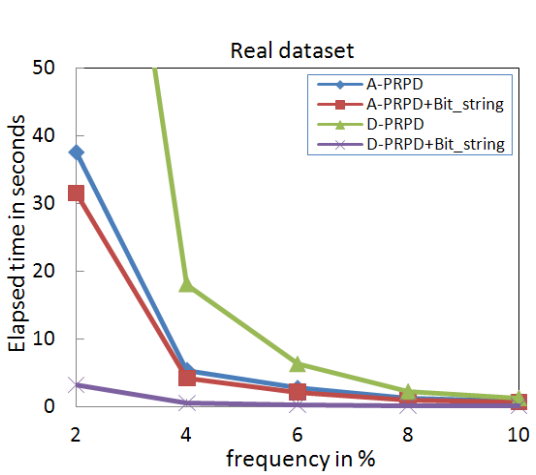
We present experimental results on the performance of our two algorithms and these two ones improved by bit-string approaches. All the experiments were conducted on a IBM desktop computer with a 2.4 Ghz Intel(R) Pentium(R) quad-core processor with 4 gigabytes main memory running Microsoft Windows XP Professional sp2 (32-bit) operating system. The algorithms were implemented in C++ with Standard Template Library (STL). The source codes of these algorithms are available at URL (http://mpc.cs.nctu.edu.tw/~stevechiu/exp_data/prpd_algo.zip). Note that the runtime was measured with the output turned off. For our experimental evaluation we used both real and

synthetic set-sequence data.

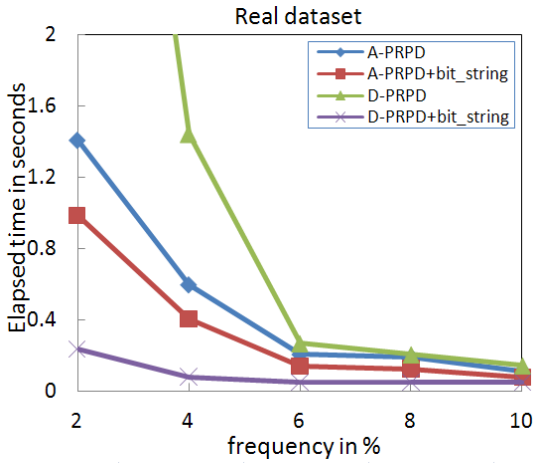
Table 4-2. Parameters of experiments

Parameter	**Description**
t	Minimal frequency threshold
S	Average size of a set-sequence data
T	Average cardinality of a set in a set-sequence data
N	Number of distinct elements in a set-sequence data

The real music objects of MIDI format were collected from internet. There were 143 music objects in total. Interested readers can download these music objects at URL (http://mpc.cs.nctu.edu.tw/~stevechiu/exp_data/music143.zip). They were classical music objects composed by various composers in different periods, from Baroque to Romantic. After preprocessing, each music object was converted into a set-sequence data. The average size of a set-sequence data was 1451 and the average cardinality of a set was 1.89. In the experiments, these set-sequences were represented in exact pitch. According to MIDI standard, the alphabet size of EPV (exact pitch value representation) was 128. We counted distinct elements of every set-sequence from 1 to 128. The real data music objects were average 46 distinct elements in a set-sequence data. As far as the synthetic dataset is concerned, the dataset is generated based on the method [18] with some modifications to generate set-sequence data and patterns. The set-sequence patterns and a set-sequence data are generated with uniform note distribution. Each generated set-sequence pattern is duplicated into several instances. These instances are inserted into the generated set-sequence data. Due to flexibility of synthetic dataset, we can control four factors which dominate the performance of the proposed algorithms in Table 4-2: minimal frequency threshold *t*, the average size of a set-sequence data $|S|$, the average cardinality of a set in a set-sequence data $|T|$ and the number of distinct elements in a set-sequence data $|N|$.



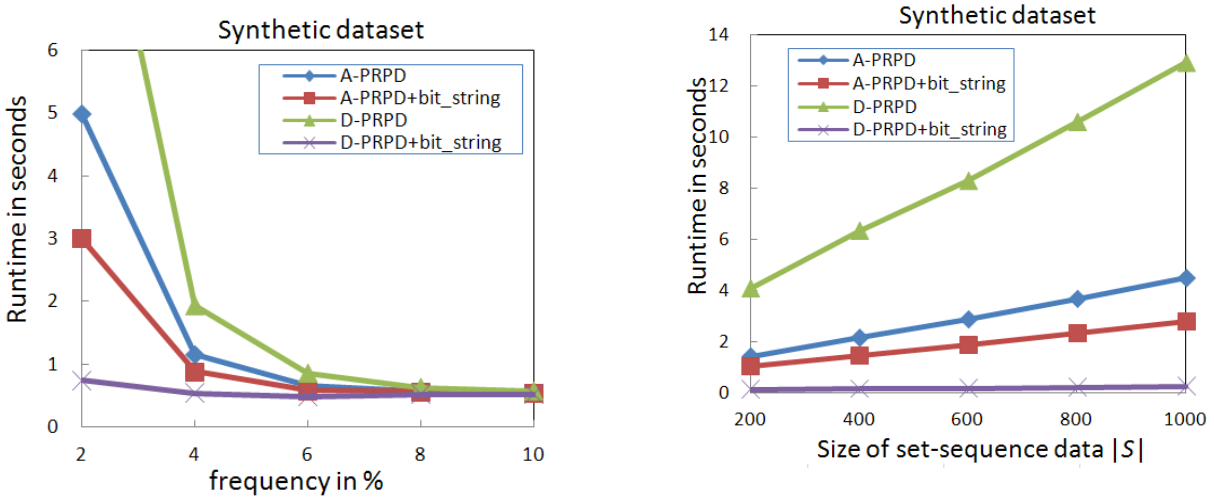
(a)



(b)

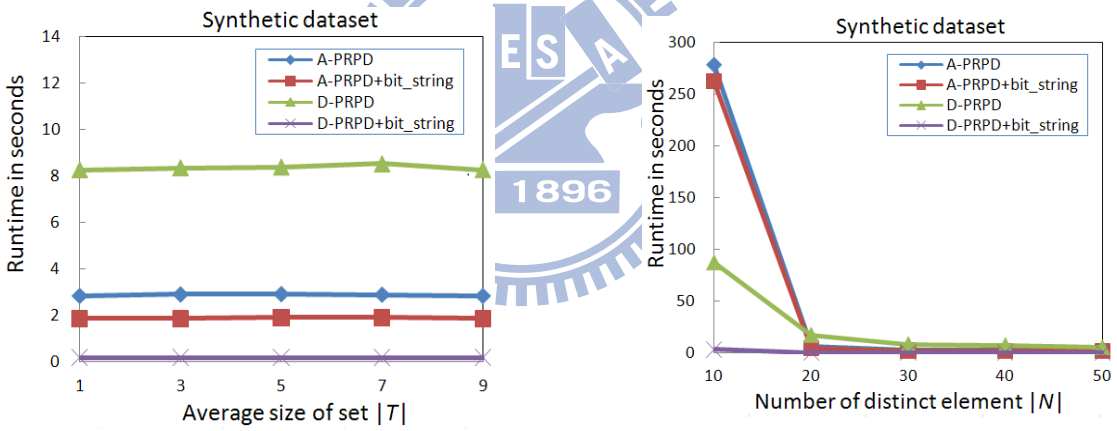
Figure 4-12. Elapsed time versus frequency for the real dataset, |*S*|:1451, |*T*|:1.89 (a) real dataset (|*N*|:46 in (PI, -)), (b) real dataset (|*N*|:72 in (EPV, EDV))

The first two experiments, as shown in Figure 4-12a and Figure 4-12b, illustrate the elapsed time of *A-PRPD*, *D-PRPD*, "*A-PRPD+bit_string*" and "*D-PRPD+bit_string*" with respect to the minimal frequency threshold *t* (percentage in total number) on real data in (PI, -) and (EPV, EDV) representations, respectively. Comparing Figure 4-12a and Figure 4-12b, we can notice that the elapsed time of all algorithms performing on real data in (PI, -) is higher than in (EPV, EDV). For most case, |*N*| in (EPV, EDV) representation is larger than in (PI, -) representation while one more attribute is considered. In the same condition, while |*N*| is lower, more polyphonic repeating patterns found leads to more candidates being checked and generated. By analyzing these two real datasets, the average maximal length of discovered repeating pattern is 15.8 and 4.7, in (EPI, -) and (EPV, EDV), respectively. Hence, the elapsed time mining in (PI, -) representation is longer than in (EPV, EDV). "*D-PRPD + bit_string*" clearly outperforms the others under two representations.



(a) (b)

Figure 4-13. Synthetic dataset: (a) elapsed time versus frequency |*S*|:1000, |*T*|:2, |*N*|:40, (b) elapsed time versus average size of a set-sequence data, *t*:4%, |*S*|:1000, |*T*|:2



(a) (b)

Figure 4-14. Synthetic dataset: (a) elapsed time versus average cardinality of a set in *sd*, *t*:4%, |*S*|:1000, |*N*|:40, (b) elapsed time versus number of distinct of elements in *sd*, *t*:4%,|*S*|:1000, |*T*|:2

Notice that *D-PRPD*, however, shows the longest elapsed time. The major diversity is caused by the large number of candidate generated by *D-PRPD*. While frequency counting operation costs time, the drawback imposes much elapsed time on *D-PRPD*. Although for most case

A-PRPD generates fewer candidates than *D-PRPD* generates, it takes too much time to check whether a candidate can be generated (see *pattern-extend* in *A-PRPD*). While string-bit approach is proposed for frequency counting in both algorithms, the algorithm taking much time for frequency counting will takes more advantages. Therefore, the bit-string approach improves much performance in *D-PRPD* than in *A-PRPD*.

For synthetic dataset, Figure 4-13 illustrates the elapsed time versus t and $|S|$ (average size of a set-sequence data *sd*). In Figure 4-13a, the elapsed time of four algorithms with respect to t for synthetic dataset is shown. The average number of found polyphonic repeating patterns at t=2%, t=4%, t=6%, t=8%, and t=10% is 597.89, 150.03, 76.83, 41.43, and 30.13, respectively. The result is similar to that of real dataset; that is, "*D-PRPD + bit_string*" is consistently the most efficient in these four experiments. For the varied $|S|$, our algorithms are linearly scalable, as Figure 4-13b shown. Especially, the algorithms with bit-string index perform in uniform behavior. Because the bit-string approach uses small storage to index the elements and count frequency by low-level binary operation, the elapsed time is almost not affected by the average size of *sd*. However, the algorithms without bit-string approaches need to check more positions for a candidate pattern as the average size of *sd* increases. In addition, Figure 4-14 illustrates $|T|$ (average cardinality of a set in *sd*) and $|N|$ (number of distinct elements in *sd*). For the varied $|T|$, the elapsed time is not affected by this factor, as Figure 4-14a shown. In Figure 4-14b, one thing needs to be noticed is that the *D-PRPD* performs well than *A-PRPD* while $|N|$ is less than about 19. This is because when $|N|$ is smaller, the number of the possible frequent length-1 patterns is fewer. That is, in *D-PRPD* fewer candidates need to check their frequency. Overall, "*D-PRPD + bit_string*" outperforms than the others in most cases.

4.5.2 Effectiveness

To show the effectiveness, we give two examples from the results after performing our

polyphonic repeating pattern algorithms. For clear readability, we only demonstrate the pattern with higher frequency.

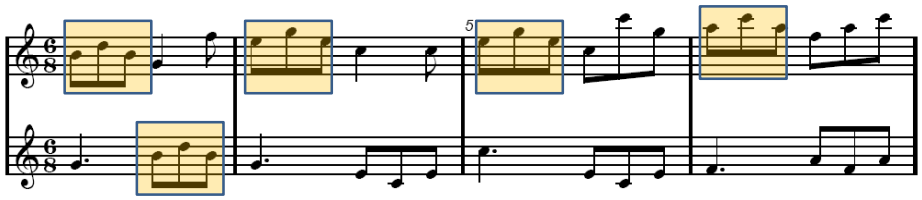


Figure 4-15. A pattern appearing in different voices is discovered from C. Nichelmann's Gigue

The first example from C. Nichelmann's "Gigue" is shown in Figure 4-15. A pattern is discovered in the music represented in $sd_{(PI,EDV)}$ and it's instances are in rectangle. While these instances of this pattern locate over different voices, our algorithms have an ability to find this pattern.

The second example is an excerpt from Chopin's Op.18, as Figure 4-16 shown. The music is represented in (PI, EDV), where PI is real transposition. We show two significant polyphonic repeating patterns discovered by our algorithm: one is in colored rectangle and the other one is in colorless rectangle. The pattern in colored rectangle is one of the important motifs in this music. According to motif development, the varied motif is transformed by shifted its original pitch. Thus, while the music is viewed in pitch interval of real transportation perspective, this kind of patterns can be found. On the other words, only the colorless pattern can be found from the music represented in (EPV, EDV). The patterns discovered from $sd_{(PI,EDV)}$ is contained by the patterns discovered from $sd_{(EPV,EDV)}$. Note that the instances in colorless rectangle show an example that this period of repetition cannot be found by traditional repeating patterns mining algorithm because the circled note could lead to the fault of the main melody extraction approach [51]. The significant motif pattern can be found when the music is in polyphonic form and discovered by our proposed algorithms.

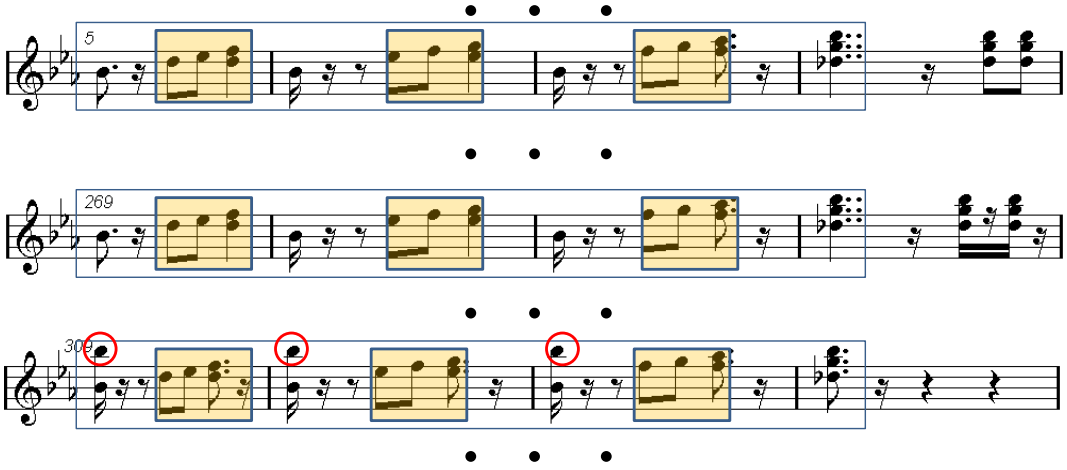


Figure 4-16. The patterns discovered from Chopin's "Grande Valse brillante" (Op. 18)

As these two examples shown, we believe that most significant musical patterns can be found. Moreover, we can find not only traditional repeating patterns in monophonic form but also more significant repeating patterns in polyphonic form which cannot be found by previous approaches. We think that there are two future directions can be made to improve our approaches. First, our developed approaches match the instances of the pattern exactly thereby missing some instances which appears approximately. Compare to approximate repeating pattern, it will be a different challenge to define what is similar in polyphonic form. Second, different patterns are discovered in different representations by our proposed approaches. An interesting direction is to design a method to present only significant patterns by integrating all results of discovered patterns in different representations.

CHAPTER 5 CONCLUSIONS AND FUTURE WORK

In this Chapter, summaries of our works are given. Some possible future works are also discussed.

5.1 Conclusions

5.1.1 Summary of Automatic Music Arrangement Framework

In this dissertation, we propose a new framework that is able to arrange multipart scores for an instrument with consideration of its role in music. The arrangement element analysis shows an important factor for arrangement, and can contribute to main melody extraction. To test our framework, we implemented a system which arranges for a piano. The Turing-test experiment shows that it is difficult to distinguish between human- and system-arranged music. While our system is able to produce viable and adaptable arrangement for piano, it can also be applied to many other instruments with the modification of playability function.

5.1.2 Summary of Polyphonic Repeating Pattern Mining

In this chapter, we studied a problem of polyphonic repeating patterns in music. A music object is modeled as a set-sequence data. We formally defined the polyphonic repeating pattern discovery problem. Two algorithms, *A-PRPD* (Apriori-based Polyphonic Repeating Pattern Discovery) and *D-PRPD* (DFS-based Polyphonic Repeating Pattern Discovery), are proposed for mining polyphonic repeating patterns from music data. *A-PRPD* uses Apriori-based method to discover longer set-sequence patterns, and *D-PRPD* maintains a lexicographic tree which provides a path to search these patterns. Furthermore, we also proposed a bit-string approach to improve the efficiency of frequency counting for both algorithms. Our experimental results demonstrate that *D-PRPD* with bit-string approach

outperforms others in most case. An interesting direction for future work is to consider how to extend these techniques in general to discover other kinds of advanced patterns.

5.2 Future Work

With the capabilities of the proposed music arrangement framework, there are several interesting extensions on this framework, as listed below.

Arranging for Various Instruments

In our proposed automatic music arrangement framework, it is able to arrange multipart scores for an instrument. Since the piano arrangement system is implemented, we will try to arrange for the other instruments by designing the various playability functions. Some playability functions are not intuitive to design, such as guitar, piano, etc. How to design these playability functions is an interesting research topic.

Arranging for an Ensemble

While the proposed framework can arrange for various instruments, the next interesting research issue is how to arrange for an ensemble, i.e., a set of instruments. The main problem is as follows: Given a set of instruments, how to decide which one plays which type of arrangement element is an interesting issue. After deciding the types of the arrangement element for each instrument, the system will be able to perform the phrase selection and finish the arrangement for an ensemble.

REFERENCES

[1] E. F. Adebiyi, T. Jiang and M. Kaufmann, "An Efficient Algorithm for Finding Short Approximate Non-tandem Repeats," Bioinformatics, Vol. 17, No. 1, pp. S5-S12, 2001.

[2] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," In *Proc. of International Conference on Very Large Data Bases*, (VLDB'94), 1994.

[3] R. Agrawal and R. Srikant, "Mining Sequential Patterns," In *Proc. of International Conference on Data Engineering*, (ICDE'95), 1995.

[4] G. Benson, "A Space Efficient Algorithm for Finding the Best Non-overlapping Alignment Score," Theoretical Computer Science, Vol. 145, No. 1&2, pp. 357-369, 1995.

[5] A. Berndt, K. Hartmann, N. Rober and M. Masuch, "Composition and Arrangement Techniques for Music in Interactive Immersive Environments," In *Proc. of Audio Mostly Conference*, 2006.

[6] B. Boser, I. Guyon and V. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," In *Proc. of the Fifth Annual Computational Learning Theory*, 1992.

[7] P. Brucker and L. Nordmann, "The k-track assignment problem," SIAM Journal of Computing, Vol. 52, pp. 97-122, 1994.

[8] E. Cambouropoulos, "The Local Boundary Detection Model (LBDM) and its Application in the Study of Expressive Timing," In *Proc. of International Computer Music Conference*, (ICMC'01), 2001.

[9] S.-C. Chiu and M.-K. Shan, "Computer Music Composition Based on Discovered Music Patterns," In *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, (SMC'06), 2006.

[10] S.-C. Chiu, M.-K. Shan, J.-L. Huang and H.-F. Li, "Mining Polyphonic Repeating Patterns from Music Data Using Bit-string Based Approaches," In *Proc. of International Conference on Multimedia and Expo*, (ICME'09), 2009.

[11] S.-Y. Chiu, S.-C. Chiu and J.-L. Huang, "On Mining Repeating Pattern with Gap Constraint," In *Proc. of International Symposium on Advances of High Performance Computing and Networking*, (AHPCN'09), 2009.

[12] J. W. Chung, "The Affective Remixer: Personalized Music Arranging," In *Proc. of Computer-Human Interaction*, (ACM SIGCHI'06), 2006.

[13] D. Conklin and C. Anagnostopoulou, "Representation and Discovery of Multiple Viewpoint Patterns," In *Proc. of International Conference on Computer Music*, (ICMC'01), 2001.

[14] D. Conklin, "Representation and Discovery of Vertical Patterns in Music," In *Proc. of International Conference on Music and Artificial Intelligence*, (ICMAI'02), 2002.

[15] V. Corozine, Arranging Music for the Real World, Mel Bay, 2002.

[16] R. Daniel and W. D. Potter, "GA-based Music Arranging for Guitar," In *Proc. of International Congress on Evolutionary Computation*, (CEC'06), 2006.

[17] T. Hastie and R. Tibshrani, "Classification by Pairwise Coupling," The Annals of Statistics, Vol. 26, No. 2, 1998.

[18] J.-L. Hsu, C.-C. Liu and A. L.-P. Chen, "Efficient Repeating Pattern Finding in Music Databases," In *Proc. of International Conference on Information and Knowledge Management*, (CIKM'98), 1998.

[19] J.-L. Hsu, C.-C. Liu and A. L.-P. Chen, "Discovering Nontrivial Repeating Pattern in Music Data," IEEE Transactions on Multimedia, Vol. 3, No. 3, pp. 311-325, 2001.

[20] J.-L. Hsu, A. L.-P. Chen and H.-C. Chen, "Finding Approximate Repeating Patterns from Sequence Data," In *Proc. of International Symposium on Music Information Retrieval*, (ISMIR'04), 2004.

[21] N. C. Jones and P. A. Pevzner, An Introduction to Bioinformatics Algorithms, The MIT Press, 2004.

[22] I. Karydis, A. Nanopoulos and Y. Manolopoulos, “Finding Maximum-length Repeating Patterns in Music Databases,” Multimedia Tools and Applications, Vol. 32, No. 1, pp. 49-71, 2007.

[23] A. A. Kasimi, E. Nechols and C. Raphael, “Automatic Fingering System (AFS),” In *Proc. of International Conference on Music Information Retrieval*, (ISMIR'05), 2005.

[24] A. A. Kasimi, E. Nechols and C. Raphael, “A Simple Algorithm for Automatic Generation of Polyphonic Piano Fingerings,” In *Proc. of International Conference on Music Information Retrieval*, (ISMIR'07), 2007.

[25] J.-L. Koh and Y.-T. Kung, “An Efficient Approach for Mining Top-K Fault-Tolerant Repeating Pattern,” In *Proc. of 11th International Conference Database Systems for Advanced Applications*, (DASFAA'06), 2006.

[26] G. Landau, J. Schmidt, “An Algorithm for Approximate Tandem Repeats,” In *Proc. of 4th Annual Symposium on Combinatorial Pattern Matching*, 1993.

[27] O. Lartillot, “Discovering Musical Patterns through Perceptive Heuristics,” In *Proc. of Internal Symposium on Music Information Retrieval*, (ISMIR'03), 2003.

[28] O. Lartillot, “A Musical Pattern Discovery System Founded on a Modeling of Listening Strategies,” Computer Music Journal, Vol. 28, No. 3, pp. 53-67, 2004.

[29] O. Lartillot, “Efficient Extraction of Closed Motivic Patterns in Multi-dimensional Symbolic Representations of Music,” In *Proc. of International Symposium on Music Information Retrieval*, (ISMIR'05), 2005.

[30] K. Lemstrom and A. Pienimaki, “On Comparing Edit Distance and Geometric Frameworks in Content-based Retrieval of Symbolically Encoded Polyphonic Music,” Musicae Scientiae, Vol. 11, pp. 135-152, 2007.

[31] C.-R. Lin, N.-H. Liu, Y.-H. Wu and A. L.-P. Chen, “Music Classification Using Significant Repeating Patterns,” In *Proc. of International Conference on Database Systems for Advanced Applications*, (DASFAA'04), 2004.

[32] N.-H. Liu, Y.-H. Wu and A. L.-P. Chen, "An Efficient Approach to Extracting Approximate Repeating Patterns in Music Databases," In *Proc. of the 10th International Conference on Database Systems for Advanced Applications,* (DASFAA'05), 2005.

[33] C.-C. Liu, J.-L. Hsu and A. L.-P. Chen, "Efficient Theme and non-Trivial Repeating Pattern Discovering in Music Database," In *Proc. of IEEE International Conference on Data Engineering,* (ICDE'99), 1999.

[34] Y.-L. Lo and W.-L. Li, "Linear Time for Discovering Non-trivial Repeating Patterns in Music Databases," In *Proc. of IEEE International Conference on Multimedia and Expo,* (ICME'04), 2004.

[35] Y.-L. Lo and C.-Y. Chen, "Fault Tolerant Non-trivial Repeating Pattern Discovering for Music Data," In *Proc. of IEEE/ACIS International Conference on Computer and Information Science,* 2006.

[36] Y.-L. Lo, W.-L. Lee and L.-H. Chang, "True Suffix Tree Approach for Discovering Non-trivial Repeating Patterns in a Music Object," Multimedia Tools and Applications, Vol. 37, No. 2, pp. 169-187, 2008.

[37] S. Lui, A. Horner and L. Ayers, "MIDI to SP-MIDI Transcoding Using Phrase Stealing," IEEE Multimedia, Vol. 13, No. 2, pp. 52-59, 2006.

[38] E. McCreight, "A Space-Economical Suffix Tree Construction Algorithm," Journal of the ACM, Vol. 23, No. 2, pp. 262-272, 1976.

[39] D. Meredith, K. Lemstrom and G. A. Wiggins, "Algorithms for Discovering Repeated Patterns in Multidimensional Representations of Polyphonic Music," Journal of New Music Research, Vol. 31, No. 4, pp. 321-345, 2003.

[40] B. Meudic, "Automatic Pattern Extraction from Polyphonic MIDI Files," In *Proc. of Computer Music Modeling and Retrieval Conference,* (CMMR'03), 2003.

[41] T. Nagashima and J. Kawashima, “Experimental Study on Arranging Music by Chaotic Neural Network,” International Journal of Intelligent Systems, Vol. 12, No. 4, pp. 232-339, 1997.

[42] B. Owsinski, The Mixing Engineer's Handbook, Thomson Course Technology, 1999.

[43] N. Patel and P. Mundur, “An N-gram based Approach for Finding the Repeating Pattern in Musical Data,” In *Proc. of European Internet and Multimedia Systems and Applications,* 2005.

[44] M. Pearce and G. Wiggins, “Towards A Framework for the Evaluation of Machine Compositions,” In *Proc. of Symposium on Artificial Intelligence and Creativity in the Arts and Sciences,* (AISB'01), 2001.

[45] N. A. Rimsky-Korsakov, Sheherazade, Op. 35 (Piano Reduction), G. Schirmer, Inc.,

[46] M. F. Sagot, “Spelling Approximate Repeated or Common Motifs Using a Suffix Tree,” Lecture Notes in Computer Science, Vol. 1380, pp. 111-127, 1998.

[47] A. Sorensen and A. R. Brown, “Introducing jMusic,” In *Proc. of the Australasian Computer Music Conference,* (interFACES), 2000.

[48] P. Spencer and P. M. Temko, A Practical Approach to the Study of Form in Music, Waveland Press, 1988.

[49] L. Stein, Structure & Style: The Study and Analysis of Musical Forms, Summy-Birchard Music, 1979.

[50] D. R. Tuohy and W. D. Potter, “An Evolved Neural Network/HC Hybrid for Tablature Creation in GA-based Guitar Arranging,” In *Proc. of International Computer Music Conference,* (ICMC'06), 2006.

[51] A. L. Uitdenbogerd and J. Zobel, “Manipulation of Music for Melody Matching,” In *Proc. of ACM International Conference on Multimedia,* (MM'98), 1998.

[52] G. White, Instrumental Arranging, McGraw-Hill, 1992.

[53] I. H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, CA: Morgan Kaufmann, 2005.

[54] Y. Yonebayashi, H. Kameoka and S. Sagayama, “Automatic Decision of Piano Fingering Based on Hidden Markov Models,” In *Proc. of International Joint Conference on Artificial Intelligence*, (IJCAI'07), 2007.

PUBLICATION LIST

Journal Papers

1. Jiun-Long Huang, **Shih-Chuan Chiu**, and Man-Kwan Shan, "Towards an Automatic Music Arrangement Framework Using Score Reduction," accepted by *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 2010.

2. **Shih-Chuan Chiu**, Hua-Fu Li, Jiun-Long Huang, and Hsin-Han You, " Incremental mining of closed inter-transaction itemsets over data stream sliding windows," accepted by *Journal of Information Science*.

3. **Shih-Chuan Chiu**, Jiun-Long Huang and Jen-He Huang, "On Processing Continuous Frequent K-N-Match Queries for Dynamic Data over Networked Data Sources," accepted by *Knowledge and Information Systems*.

4. Jiun-Long Huang, **Shih-Chuan Chiu** and Xin-Mao Huang, "GPE: A Grid-based Population Estimation Algorithm for Resource Inventory Applications over Sensor Networks," *Journal of Information Science and Engineering*, Vol. 25, No. 1, 210-218, January 2009.

Conference Papers

1. **Shih-Chuan Chiu**, Man-Kwan Shan and Jiun-Long Huang, "Automatic System for the Arrangement of Piano Reductions," *AdMIRe: International Workshop on Advances in Music Information Research 2009 (in conjunction with IEEE International Symposium on Multimedia 2009)*, 459-464, December 14 - December 16, 2009, San Diego, USA.

2. **Shih-Chuan Chiu**, Man-Kwan Shan, Jiun-Long Huang and Hua-Fu Li, "Mining Polyphonic Repeating Patterns from Music Data Using Bit-String Based Approach," *IEEE International Conference on Multimedia & Expo (ICME 2009)*, 1170-1173, June 28 - July 3, 2009, New York, USA.

3. Shin-Yi Chiu, **Shih-Chuan Chiu** and Jiun-Long Huang, "On Mining Repeating Pattern with Gap Constraint," *International Symposium on Advances of High Performance Computing and Networking (AHPCN-09)*, 557-562, June 25 - June 27, 2009, Seoul, Korea.

VITA

Shih-Chuan Chiu (邱士銓) was born on January 17, 1980 in Kaohsiung, Taiwan, Republic of China. He received the B.S. degree in Computer Science and Information Engineering from Tamkang University (TKU) and the M.S. degree in Computer Science from National Chengchi University (NCCU), in 2002 and 2004, respectively. He is currently working towards the Ph.D. degree in the Department of Computer Science at National Chiao Tung University (NCTU). His current research interests include data mining, computer music and mobile computing.