Towards an Automatic Music Arrangement Framework Using Score Reduction

JIUN-LONG HUANG and SHIH-CHUAN CHIU, National Chiao Tung University
MAN-KWAN SHAN, National Chengchi University

Score reduction is a process that arranges music for a target instrument by reducing original music. In this study we present a music arrangement framework that uses score reduction to automatically arrange music for a target instrument. 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 transform the music arrangement into an optimization problem and propose a phrase selection algorithm. 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. An approach similar to Turing test is used to evaluate the quality of the music arranged by our system. The experiment results show that our system is able to create viable music for the piano.

Categories and Subject Descriptors: H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing—*Methodologies and techniques, modeling, systems*; J.5 [**Computer Application**]: Arts and Humanities—*Fine arts, Performing arts*

General Terms: Theory

Additional Key Words and Phrases: Score reduction, automatic music arrangement, piano reduction, phrase selection

ACM Reference Format:

Huang, J.-L., Chiu, S.-C., and Shan, M.-K. 2012. Towards an automatic music arrangement framework using score reduction. ACM Trans. Multimedia Comput. Commun. Appl. 8, 1, Article 8 (January 2012), 23 pages.
DOI = 10.1145/2071396.2071404 http://doi.acm.org/10.1145/2071396.2071404

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

Authors' addresses: J.-L. Huang (corresponding author), Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, ROC; email: jlhuang@cs.nctu.edu.tw; S.-C. Chiu, Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, ROC; M.-K. Shan, Department of Computer Science, National Chengchi University, Taipei 116, Taiwan, ROC.

DOI 10.1145/2071396.2071404 http://doi.acm.org/10.1145/2071396.2071404

transcription is necessary to adapt the song for the target instrument(s) [Corozine 2002]. 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 ringtone 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) [Lui et al. 2006]. 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 stated problems.

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 focuses only 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. In this paper, 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 into consideration the characteristics and the inherent restrictions of the target instrument, such as pitch range and polyphonic limitation. Hence, 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. Piano reduction 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 [Rimsky-Korsakov 1888].

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 in this paper 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 Owsinski [1999], there are five types of arrangement elements: *lead*, *foundation*, *rhythm*, *pad*, and *fill*. Interested reader can refer to Section 2.1 for more discussions about the arrangement elements.

In this article, we propose a framework that arranges a piece of music by reducing the multipart score for a given instrument. 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). Figure 2 presents an example of the change in musical content during the framework's arrangement process. Because the arrangement elements of some instruments may vary in different sections, the music object is first divided into several segments called *segmented tracks* in *track segmentation* phase. Next, in the *arrangement*

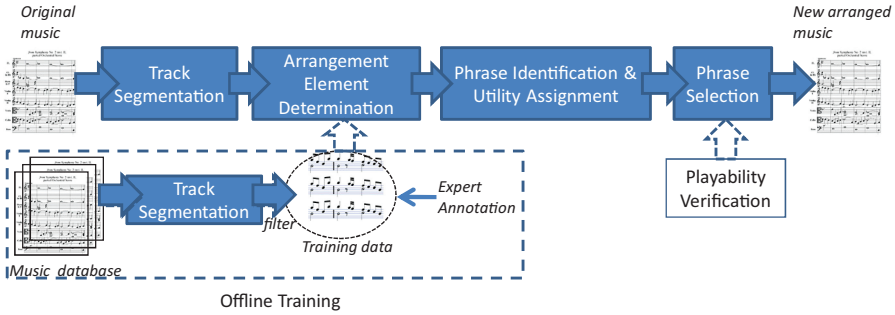

Fig. 1. The flowchart of the proposed music arrangement framework.

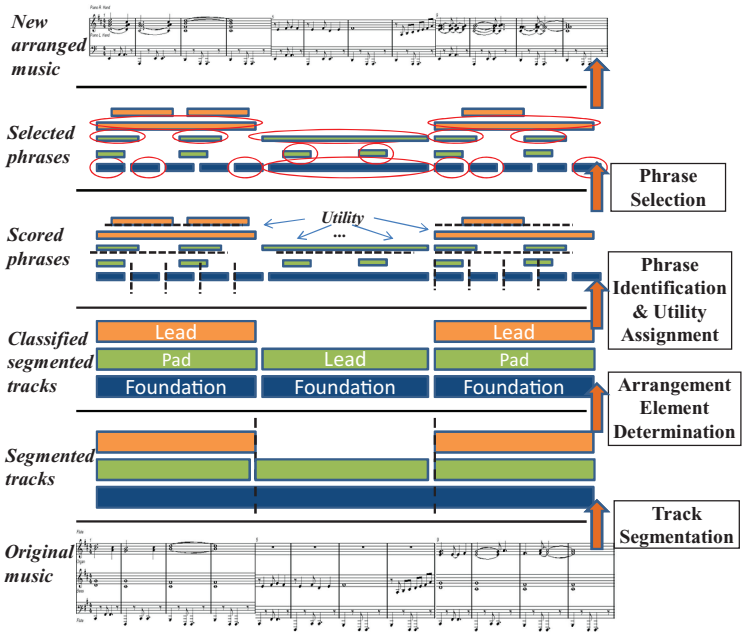

Fig. 2. An example process of the proposed music arrangement framework.

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

The remainder of this article is organized as follows. Section 2 gives a preliminary of the arrangement elements and a brief survey of related work. Section 3 gives an introduction of each component

of the proposed framework. The experimental results are given in Section 4 while Section 5 concludes this article.

2. PRELIMINARY

2.1 Arrangement Element

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 [Owsinski 1999]. 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 Owsinski [1999], 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 paper, "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

Many works related to music arrangement focus on how to transform original music by changing metainformation (tempo, timbre, etc.) or content (insert note, change pitch, re-assemble music segments, etc.) [Miranda 2001]. Nagashima and Kawashima [1997] employed chaotic neural networks to create variations on melodies. 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 et al. [2006] presented the strategies to synchronize and adopt the game music with player interaction behaviors. 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 [2006] proposed a real-time music-arranging system that reacts to the affective cues from a listener. 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,[1] 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 [Tuohy and Potter 2006], Daniel and Potter [2006] presented an approach for guitar arrangement. 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.

3. PROPOSED FRAMEWORK

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 nonsilence beats in measure i at track t, $NumTrack$ is the number of tracks, and $BeatPerMeasure$ is beats per measure.

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, $Sim_{i,i+1} < \tau$, it means that there must be at least a half number of instruments switched.

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 Section 2.1, 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*).

[1]http://www.finalemusic.com.

Table I. 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.

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.

According to the descriptions of the arrangement elements in [Owsinski 1999], 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 I.

The classifier is trained using manual tagged data for each segmented track, that is, 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.

3.3 Phrase Identification and Utility Assignment Phase

3.3.1 *Phrase Identification.* In this section, we attempt to identify the phrases from a segmented track. As mentioned in Stein [1979], 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 in Lui et al. [2006] 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 Lui et al. [2006] 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) [Cambouropoulos 2001] 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 $\{pitch, ioi, rest\}$, i $\{1, 2, \ldots, n\}$ 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 Cambouropoulos [2001] (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.

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

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

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 ϕ_{ae} (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

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

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 is defined as

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

where α_1, α_2 [0, 1]; $\alpha_1 + \alpha_2 = 1$; and $Length(phr_{st,i})$ 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 1 *(Phrase selection problem)*. Given a set of phrases, denoted as $PSet = \{p_1, p_2, \ldots, p_n\}$ 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 [Brucker and Nordmann 1994]. 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 [Brucker and Nordmann 1994] for the k-track assignment problem. Unfortunately, it is difficult to perform such integration since the algorithm proposed in Brucker and Nordmann [1994]

is optimized for the k-track assignment problem. Let's consider another problem, the exon chaining problem [Jones and Pevzner 2004], which is a special case of the k-track assignment problem with $k = 1$. Due to the simplicity of the algorithm proposed in Jones and Pevzner [2004], 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 3.4.3.

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 here.

—*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) [White 1992]. 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.

—*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 in Appendix A 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.

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 subproblems. Thus, the phrase selection algorithm is designed in a divide-and-conquer manner.

Algorithm Overview. The details of the proposed phrase selection algorithm are shown in Figure 3. 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

```
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, 2n − 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;
```

Fig. 3. Phrase selection algorithm.

phrases, and will still keep the overlap relationships between each pair of phrases.[2] There are $2n$ 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, $Opt(0,0,2n-1, MOP|\{\})$, is called to compute the optimal selection $Opt(0,0,2n-1, MOP|\{\}).sel$ and the optimal utility $Opt(0,0,2n-1, MOP|\{\}).ut$ (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 2. $Opt(bv, si, ei, room \mid CP\_List)$ 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 $i \leq j$.

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. Function Opt is designed in a divide-and-conquer manner. That is,

[2]Interested readers can refer to Jones and Pevzner [2004] for the details of the transformation.

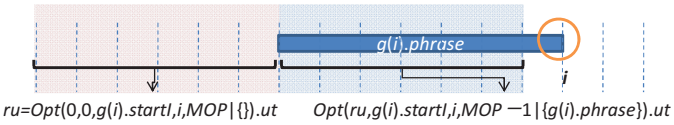

Fig. 4. An illustration of $CP\_List = \{\}$.

the whole problem can be divided into several subproblems, 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.

A phrase is selected only when the following two conditions are satisfied. The first condition is the *playable* condition (line 4) that $CP\_List \bigcup \{g(i).phrase\}$ 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 ($g(i).phrase \neq$null and ($isPlayable(CP\_List \bigcup \{g(i).phrase\}) =$ 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, $w > Opt(bv, si, i-1, room \mid CP\_List)$, where the calculation of w will be described later. If the above two conditions are satisfied, $g(i).phrase$ is selected. The optimal selection $Opt(bv, si, i, room \mid CP\_List).sel$ is updated according to the optimal selection during computing w, and the optimal utility $Opt(bv, si, i, room \mid CP\_List).ut$ is set to w. Otherwise, the optimal selection and the utility of the optimal selection are inherited from the previous results, $Opt(bv, si, i-1, room \mid CP\_List).sel$ and $Opt(bv, si, i-1, room \mid CP\_List).ut$, respectively.

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 4, the influence region of selecting $g(i).phrase$ is the region that $g(i).phrase$ locates, that is, 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, $Opt(ru, g(i).startI, i, MOP-1 \mid \{g(i).phrase\})$, 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., $ru = Opt(0, 0, g(i).startI, MOP \mid \{\}).ut$). Thus, the utility of the optimal selection when $g(i).phrase$ is selected is $w = Opt(ru, g(i).startI, i, MOP-1 \mid \{g(i).phrase\}).ut + g(i).utility$. 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, $w > Opt(bv, si, i-1, room \mid CP\_List).ut$ (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 5, when CP_List is not empty, many subregions 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 $CP\_List \bigcup \{g(i).phrase\}$. Without loss of generality, the phrases in $CP\_List \bigcup \{g(i).phrase\}$ are sorted by their start positions in ascending order and relabeled as $\{j_1, j_2, \ldots, j_{phrase_n}\}$, where $j_1.startI \leq j_2.startI \leq \cdots \leq j_{phrase_n}.startI \leq i$, 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 $Opt(0,0,j_1.startI,MOP \mid \{\}).ut$. For the second sub-region from index $j_1.startI$ to $j_2.startI$, the utility of the optimal selection of the first

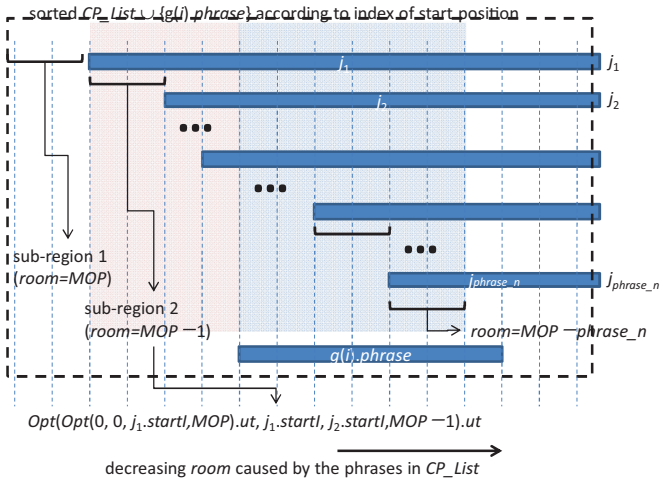

Fig. 5. An illustration of $CP\_List \neq \{\}$.

sub-region ru_0 is taken as the base value and the maximal number of overlapping phrases allowed is $MOP - 1$. Thus, $Opt(ru_0, j_1.startI, j_2.startI, MOP - 1 \mid \{j_1\})$ 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 subregion as the base value and calculate the optimal utility in this subregion by invoking function Opt in a similar manner. This process repeats until the last sub-region from $j_{phrase\_n}.startI$ to i under $MOP - phrase\_n$ has been processed by function Opt. This recurrence relation is shown as follows.

Initial condition:
$Opt(bv, si, si, room|CP\_List).ut = bv$;
Recurrence relation:
$Opt(bv, si, i, room|CP\_List).ut$

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

According to the above recurrence relation, we can notice that the functions with the same parameter (for example, $Opt(0, 0, i, MOP \mid \{\})$, 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 $room$.

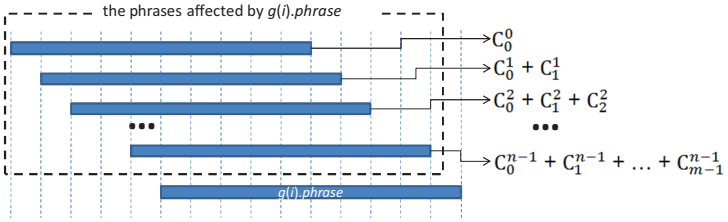

Fig. 6. An illustration of the computation at the worst case.

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 Jones and Pevzner [2004]. 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 subregions with smaller values of *room*. Since the value of *room* of each invocation of function *Opt* on each subregion 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.

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 $1 \leq i \leq 2n-1$). 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 6. 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 $C_0^{i-1} + C_1^{i-1} + \cdots + C_{m-1}^{i-1} = \sum_{j=0}^{m-1} C_j^{i-1}$, 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 $C_0^0 +$

$(C_0^1 + C_1^1) + (C_0^2 + C_1^2 + C_2^2) + \cdots + (C_0^{n-1} + C_1^{n-1} + \cdots + C_{m-1}^{n-1}) = \Phi$. And $\Phi = \sum_{j=0}^{m-1} C_j^0 + \sum_{j=0}^{m-1} C_j^1 + \cdots + \sum_{j=0}^{m-1} C_j^{n-1} \leq \sum_{j=0}^{m-1} C_j^{n-1} \sum_{j=0}^{m-1} C_j^{n-1} + \cdots + \sum_{j=0}^{m-1} C_j^{n-1} = (n-1) \times (C_0^{n-1} + C_1^{n-1} + \cdots + C_{m-1}^{n-1}) \leq (n-1) \times (C_{m-1}^{n-1} + C_{m-1}^{n-1} + \cdots + C_{m-1}^{n-1}) = (n-1) \times m \times C_{m-1}^{n-1} \leq (n-1) \times m \times n^m = O(mn^m)$. Because the playability function is performed when each phrase is selected, the time complexity of the phrase selection algorithm at the worst case is $O(\Psi mn^m)$. 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 Appendix D for the execution time of the proposed algorithm on several real cases.

4. EXPERIMENTAL RESULTS

Our music arrangement system was implemented in Java, along with two open source packages, jMusic [Sorensen and Brown 2000] and Weka [Witten and Frank 2005]. 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/).

4.1 Effectiveness of Arrangement Element Determination

In implementing of the arrangement element determination, we chose the support vector machine (SVM) [Boser et al. 1992] 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 multiclass 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 multiclass case, Hastie and Tibshirani's pairwise coupling method [Hastie and Tibshrani 1998] 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.

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

Table II. Confusion Matrix for Five Arrangement Elements with Tenfold Cross-Validation

	Classifier As				
Arrangement Element	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

these parameters are listed in Table VI in Appendix C). The confusion matrix of classification result is shown in Table II. The f-measures for *foundation*, *rhythm*, *pad*, *lead*, and *fill* are 0.907, 0.826, 0.72, 0.813, and 0.4 respectively.

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.

4.2 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 Pearce and Wiggins [2001] proposed a method to evaluate the computer music composition system. 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 VII in Appendix C. The same setting was also adopted in the succeeding experiments. The experiment used the first 16 music objects in Table III. The music objects were sorted randomly and displayed to the subjects on the Web page.[3] 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.

The results are shown in Table IV. 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

[3]http://www.cs.nctu.edu.tw/~scchiu/mas/survey.html.

Table III. 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
Violin Sonata No. 3, Mov. 1 (S7,A7)	Handel	S
Love You to (P3)	George Harrison	H
Ring Ring de Banjo (P4)	Stephen C. Foster	H
Goober Peas (P5)	N/A (USA Civil War)	H

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).

Table IV. 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*.

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.

4.3 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 III, 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

Table V. 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

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.

5. CONCLUSION

We present a framework that arranges 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 in our framework.

```
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;
```

Fig. 7. Piano-Right-Hand-Playability function.

APPENDIXES

APPENDIX A. 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 in Kasimi et al. [2005, 2007] and Yonebayashi et al. [2007]; however, the work cannot be used to determine whether a piece of music can be played by piano. We refer to White [1992] 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 7. 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 8, 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

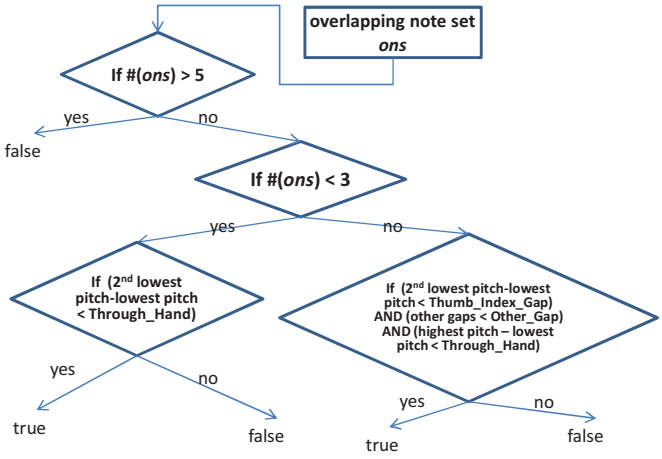

Fig. 8. Finger-Assignable flowchart.

users according to the size of their hands. Finally, the set of overlapping phrases is playable since all *ons* are assignable for fingers.

APPENDIX B. Running Example of Phrase Selection Algorithm

Figure 9 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 9(a) shows the identified phrases in the score; and Figure 9(b) shows the phrases with utilities represented by weighted intervals. Figure 9(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 $p3$, and thus we have $g(6).phrase = p3$, $g(6).utility = 2$ 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 9(e), after $Opt(0,0,13,2\ |\{\})$ is finished, $Opt(0,0,13,2\ |\{\}).ut = 20$ and $Opt(0,0,13,2\ |\{\}).sel = \{p0, p1, p2, p6\}$ are returned as the result of the proposed algorithm.

Here we use the example in Figure 9 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 $p0$. Due to the reason that phrase $p0$ is playable and it is allowed (room > 0) to select $p0$, $p0$ is selected and the function $Opt(0,0,3,1|\{p0\})$ is called to check if there is any influence of selecting $p0$. No other phrases can be seen from index 0 to 3, and thus, $p0$ can be selected ($Opt(0,0,3,2\ |\{\}).sel = \{p0\}$) at this moment.

While determining whether phrase $p3$ (i.e., $g(6).phrase$) is worth to be selected, $p3$ is selected first and $Opt(0,1,5,1\ |\{p3\})$ is called. In solving $Opt(0,1,5,1\ |\{p3\})$, it meets $p1$ and $p2$ because $g(3).phrase$ and $g(5).phrase$ are not null. However, $p0$ (i.e., $g(3).phrase$) and the phrase in the CP_List (i.e., $p3$), are

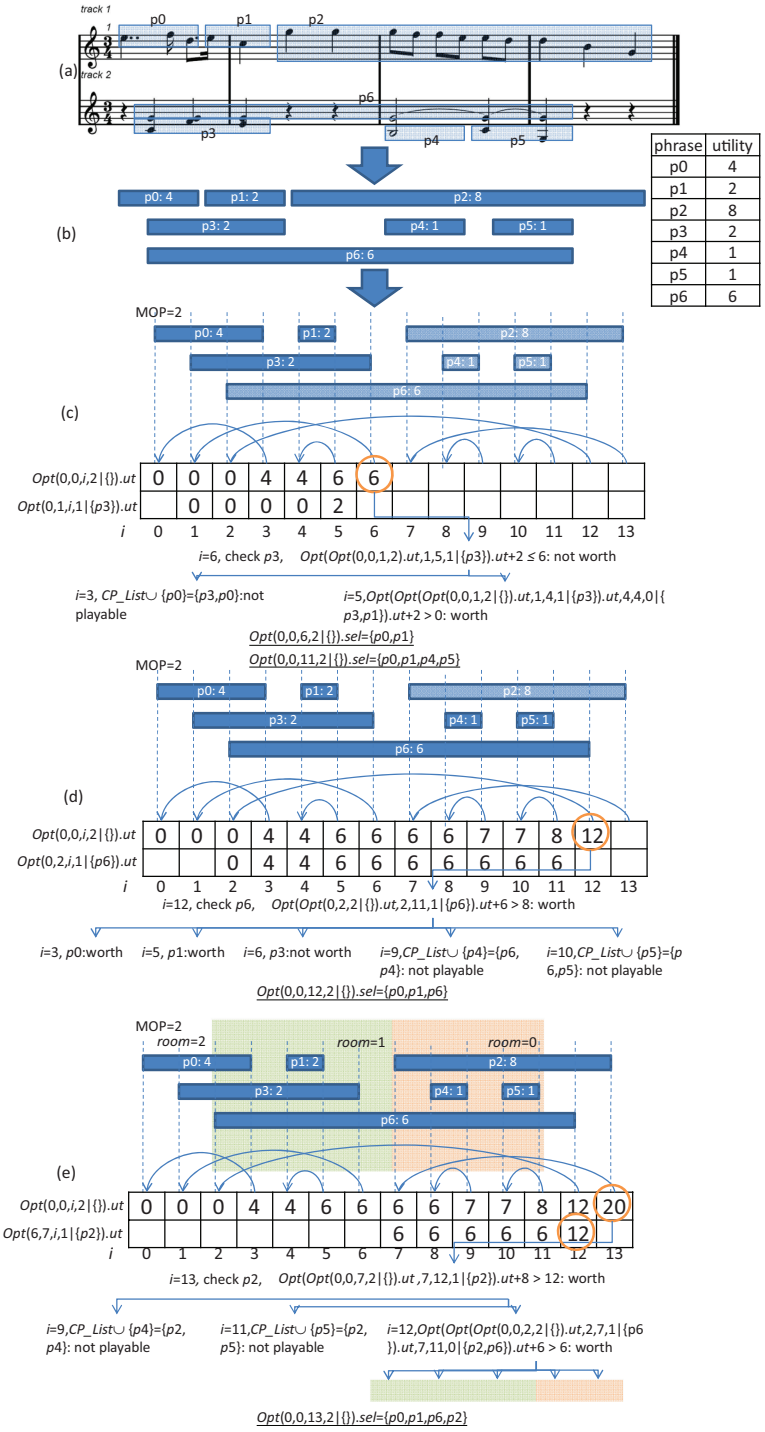

Fig. 9. An example of the phrase selection algorithm: (a) the identified phrases in the given score; (b) the identified phrases represented by intervals; (c) a snapshot at index 6; (d) a snapshot at index 12; (e) a snapshot at index 13.

Table VI. 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

not playable due to the reason that the overlapping part of phrase $p0$ and $p3$ (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 $p3$ is selected. However, the utility of the optimal selection when $p3$ is selected is not worthier than the utility when $p3$ is not selected (i.e., $Opt(0,1,5,1|\{p3\}).ut + g(6).utility = 4 < Opt(0,0,5,2\ |\{\}).ut = 6$). Thus, $Opt(0,0,6,2\ |\{\}).ut$ is set to 6. As shown in Figure 9(c), the optimal selection $Opt(0,0,6,2\ |\{\}).sel$ is $\{p0,p1\}$. When function *Opt* goes to index 12, the utility of the optimal selection when $p6$ is selected (i.e., $Opt(Opt(0,0,2,2\ |\{\}).ut,2,11,1|\{p6\}).ut+6 = 12$) is worthier than the utility when $p6$ 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 $\{p0,p1,p6\}$.

Consider the example that phrase $p2$ is checked in Figure 9(e). Function $Opt(6,7,12,1 \mid \{p2\})$ is called to examine the influence of the optimal selection under the condition that $p2$ is selected. During the process in $Opt(6,7,12,1 \mid \{p2\})$, $p4$, $p5$ and $p6$ will be examined sequentially. Phrase $p4$ and $p5$ are not playable with the phrase in CP_List (i.e., $p2$), while $p6$ is playable with the phrase in CP_List and $room > 0$ (the value of *room* is 1). Now function *Opt* examines whether $p6$ is worth to be selected by considering $p6$ with CP_List. At this moment, $CP\_List \cup g(12).phrase$ contains two phrases ($p2$ and $p6$). Then, $p2$ and $p6$ are sorted and relabeled according to their start positions. Thus, $j_1 = p6$ and $j_2 = p2$. 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 $Opt(0,2,7,1 \mid \{p6\})$ for computing the optimal utility of the second sub-region (from 2 to 7 with *room* $= 1$). After obtaining $Opt(0,2,7,1 \mid \{p6\}).ut = 6$, 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 $Opt(6,7,11,0 \mid \{p6,p0\})$. Since phrase $p6$ is worth to be selected under the condition that phrase $p2$ is selected ($w > 6$), $p2$ is selected. Back to the outermost *Opt*, phrase $p2$ is also worth to be selected. Thus, the final optimal selection is $\{p0, p1, p6, p2\}$ and the utility of the optimal selection is 20.

APPENDIX C. Parameters in the Piano Arrangement System

We put the parameters of the SVM classier and our piano arrangement system in Table VI and Table VII, respectively.

APPENDIX D. 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 VIII. 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 VII. 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.α_1, *UA.*α_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.

Table VIII. 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	57.702 s
Lilacs *3	1:29	6	310	11.64	9552	42.817 s
Sym. No. 5 *4	1:05	17	112	32.28	38918	156.663 s

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

APPENDIX E (GLOSSARY)

Arrangement element. Arrangement element is the function of a piece of music performed by an instrument.

Phrase. A phrase is continuous line of notes. It can be a melody, countermelody, or an important accompanying line. [Liu et al., 2006]

Arrangement. The term arrangement is often considered synonymous with transcriptions. But, in computer music field, transcription is commonly used to mean the act of notating a piece or a sound which was previous annotated. Thus, we still use the term, arrangement, to make a distinction.

Score reduction. Score reduction is a process that arranges music for a target instrument by reducing original music.

REFERENCES

BERNDT, A., HARTMANN, K., ROBER, N., AND MASUCH, M. 2006. Composition and arrangement techniques for music in interactive immersive environments. In *Proceedings of the Audio Mostly Conference*.

BOSER, B., GUYON, I., AND VAPNIK, V. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*.

BRUCKER, P. AND NORDMANN, L. 1994. The k-track assignment problem. *SIAM J. Comput.* 52.

CAMBOUROPOULOS, E. 2001. The local boundary detection model (LBDM) and its application in the study of expressive timing. In *Proceedings of the International Computer Music Conference (ICMC'01)*.

CHUNG, J. W. 2006. The affective remixer: Personalized music arranging. In *Proceedings of the Conference on Human Factors in Computing Systems (ACM SIGCHI'06)*.

COROZINE, V. 2002. *Arranging Music for the Real World.* Mel Bay.

DANIEL, R. AND POTTER, W. D. 2006. GA-based music arranging for guitar. In *Proceedings of the International Congress on Evolutionary Computation (CEC'06)*.

HASTIE, T. AND TIBSHRANI, R. 1998. Classification by pairwise coupling. *Ann. Stat. 26*, 2.

JONES, N. C. AND PEVZNER, P. A. 2004. *An Introduction to Bioinformatics Algorithms*. MIT Press.

KASIMI, A. A., NECHOLS, E., AND RAPHAEL, C. 2007. A simple algorithm for automatic generation of polyphonic piano fingerings. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR'07)*.

KASIMI, A. A., NECHOLS, E., AND RAPHAEL, C. 2005. Automatic fingering system (AFS). In *Proceedings of the International Conference on Music Information Retrieval (ISMIR'05)*.

LUI, S., HORNER, A., AND AYERS, L. 2006. MIDI to SP-MIDI transcoding using phrase stealing. *IEEE Multimedia 13*, 2.

MIRANDA, E. R. 2001. *Composing Music with Computers*. Focal Press.

NAGASHIMA, T. AND KAWASHIMA, J. 1997. Experimental study on arranging music by chaotic neural network. *Int. J. Intell. Syst. 12*, 4.

OWSINSKI, B. 1999. *The Mixing Engineer's Handbook*. Thomson Course Technology.

PEARCE, M. AND WIGGINS, G. 2001. Towards a framework for the evaluation of machine Compositions. In *Proceedings of the Symposium on Artificial Intelligence and Creativity in the Arts and Sciences (AISB'01)*.

RIMSKY-KORSAKOV, N. A. 1888. *Sheherazade, Op. 35* (Piano Reduction). G. Schirmer, Inc.

SORENSEN, A. AND BROWN, A. R. 2000. Introducing jMusic. In *Proceedings of the Australasian Computer Music Conference*.

STEIN, L. 1979. *Structure & Style: The Study and Analysis of Musical Forms*. Summy-Birchard Music.

TUOHY, D. R. AND POTTER, W. D. 2006. An evolved neural network/HC hybrid for tablature creation in GA-based guitar arranging. In *Proceedings of the International Computer Music Conference (ICMC'06)*.

WHITE, G. 1992. *Instrumental Arranging.* McGraw-Hill.

WITTEN, I. H. AND FRANK, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

YONEBAYASHI, Y., KAMEOKA, H., AND SAGAYAMA, S. 2007. Automatic decision of piano fingering based on hidden Markov models. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'07)*.

Received June 2010; revised November 2010; accepted November 2010