Evolutionary Interactive Music Composition

Dao-yung Fu
Tsu-yu Wu
Chin-te Chen
Kai-chu Wu
Ying-ping Chen

NCLab Report No. NCL-TR-2006001
January 2006

Natural Computing Laboratory (NCLab)
Department of Computer Science
National Chiao Tung University
329 Engineering Building C
1001 Ta Hsueh Road
HsinChu City 300, TAIWAN
http://nclab.tw/

Evolutionary Interactive Music Composition

Dao-yung Fu, Tsu-yu Wu, Chin-te Chen, Kai-chu Wu, and Ying-ping Chen
Department of Computer Science
National Chiao Tung University
HsinChu City 300, Taiwan
{tyfu, tywu, chinte, kcwu, ypchen}@cs.nctu.edu.tw

January 30, 2006

Abstract

This paper proposes and describes the CFE framework—Composition, Feedback, and Evolution—and presents an interactive music composition system. The system composes short, manageable pieces of music by interacting with users. The most important features of the system include creating customized music according to the user preference and the facilities specifically designed for producing massive music. We present the structure as well as the implementation of the system and the auxiliary functionalities that enhance the system. We also introduce the auto-feedback test with which we verify and evaluate the interactive music composition system, followed by the discussion and conclusions.

1 Introduction

Music plays an important role in our daily life. It makes us sad, happy, and excited. The definition of pleasant music is different from people to people; some people love classic music, and some love heavy metal. We can easily observe that almost everyone puts different ringtones on their cellular phones. As a result, customization for pleasant music is desirable for our modern life, just like a customized suit is fitter than a T-shirt with only S, M, and L sizes. In addition to customization, massive music is needed for some applications, such as the scene music of games and the background music of web pages. It would be fantastic if common people can create music on their own. Although there are lots of computer software which can help people compose music, it is still hard, if not impossible, to create pleasant music for unskilled people. Hence, we are trying to make the computer automatically create music for us instead of merely letting us put notes into tracks.

In order to reach the goal, making the computer automatically create music, we develop a system which creates music by interacting with users. The created music can be used on cellular phones, alarms, or other devices of which the sound of music can be set or load by the user. We design the system according to two backgrounds. One is Evolutionary Computation [1, 2, 3, 4, 5] . Based on the concept of Evolutionary Computation, we build our kernel algorithm. The other background is the MIDI format. When we create music in the MIDI format, we can guarantee that the created music can be played on computers, cellular phones, or other customizable electronic devices. The mankind needs a system to sing out their voice of soul. Therefore, we try to do our best to approach this goal in this study.

The organization of the paper is as follows. Section 2 reviews the state of the art of creating music in the field of evolutionary computation. Section 3 describe our CFE framework in

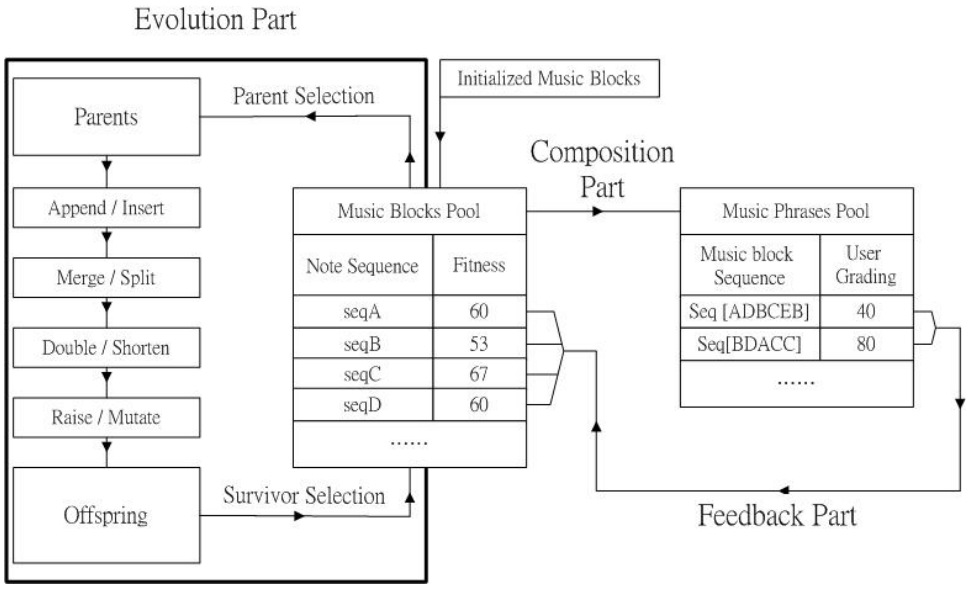

Figure 1: The structure of the CFE framework.

detail, and section 4 presents the system implementation and the auxiliary functionalities for enhancement. Section 5 describes the auto-feedback test with which we verify and evaluate the music composition system. We discuss this research work in section 6. Finally, section 7 concludes this paper.

2 State of the Art

There have been many studies attempting to compose music with the techniques of evolutionary computation. These studies vary a lot in their adopted evolutionary techniques. We can broadly classify these research works by analyzing the three stages in their evolutionary processes: initialization stage, evolution stage, and grading stage.

The initialization stage: It is about the methods which are used to initialize the population of the evolutionary environment. There are many kinds of initialization procedure in the literature. Method 1, the random initialization [6], provides a relatively worse quality for the initial individuals but is limited by fewer restrictions than other methods. Method 2, the complex function initialization [6, 7], initializes the population through certain pre-designed rules and only produces individuals which satisfy the specified restrictions. Method 3, the song initialization [8], initializes the population by analyzing one or more available songs and by decomposing these songs into individuals.

The evolution stage: It is related to the focus of evolutionary domains and the genotype of music presentation. For example, in some studies, the genotype of music is a sequence of notes [4, 8] or a sequence of functions, such as $sin(\cdot)$ and/or $cos(\cdot)$ which are used to produce a part of the generated music [7]. Their evolutionary domains focus on the theme of music. In other studies, the genotype of music is a sequence of tempo numbers [9], and it only concentrates on the tempo of music.

The grading method: It is about how to judge the music composed by evolutionary systems. One way is to judge the music with the real audience through either real-time judging [4, 6] or non-real-time judging [6] methods. No matter with what kind of judging methods, it

may need lots of judging runs. Another way is to use the neural network evaluation function [6]. This method needs to produce the neural network module, and it takes tremendous time to train the module. The other way is to utilize some fitness functions [4, 6, 7]. Constructing a function to appropriately evaluate the generated music is critical in these approaches.

3 The CFE Framework

The evolutionary algorithms may be an suitable technique to optimize music if we take the user preference as the fitness of the environment, like many of the previous studies reviewed in section 2. However, what if the user needs more than one song? Similar jobs should be done over and over to produce more songs which the user favors. To avoid this awkward situation and to achieve a better music composition system, we propose the CFE framework in this section.

3.1 Introduction to the CFE Framework

The CFE framework contains three major parts, Composition, Feedback, and Evolution. In this framework, we try to find the fittest way to compose music rather than the fittest melody for the user. To be more accurate, the individuals in the evolutionary environment is no longer complete songs but some musical elements or guidelines. The Composition part uses these musical elements and guidelines to construct new melodies. The composed melodies then wait for the user's response such as making a grade or just telling good or bad. After the system receives the information, the Feedback part distributes these feedbacks among the musical elements and guidelines to evaluate how fit these composing components are. For discovering better methods, the technique of evolutionary computation is adopted such that new elements and guidelines are born into the population.

The three parts can be done independently. Therefore, once the user is satisfied with the composed music, no more work is necessary when he or she needs more pieces of music because Composition can be conducted alone. Since Composition and Evolution are isolated, for making use of the domain knowledge, such as the constraints, indication, and implications in the music theory, it is easier to embed such knowledge into the Composition part than to interfere with the regular operations of evolutionary algorithms. The separation of Feedback and Evolution provides the feature that the pace of evolution can be determined by how many feedbacks we get from the user.

The CFE framework will be described in detail in the following sections. Figure 1 shows the design of the system implemented in the present work.

3.2 Design of the Composition Part

In the present work, the type of music which we focus on is the theme music with a specific length, say, 8 measures or 16 measures, named *music phrases*. Inspired by some pop music that some subsequences appearing in a song frequently and repeatedly, we take a layered approach to find out the potentially good sequences of notes. Our system deals with the short theme music by using two levels of hierarchy. The music phrase consists of the variable-length small sequences of notes, called *music blocks*. Composition picks the favored music blocks and fill in the incomplete music phrases until the specified length is reached.

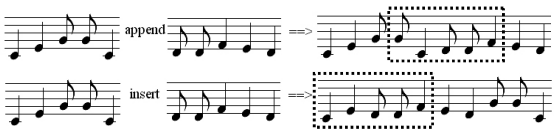

Figure 2: Operations: Append and Insert.

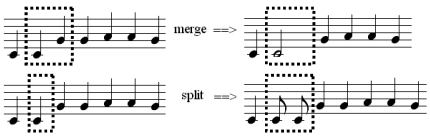

Figure 3: Operations: Merge and Split.

3.3 Design of the Feedback Part

The design of the Feedback part provides the interface for users to make their responses to the system. We simply let users listen to the music phrase composed by Composition and let them grade it in the range from 0 to 100. It is not too complex for common users because the grading is episodic such that users do not have to listen to the music nervously for the need to make real-time responses like applauding. Once the grading is made, the score is distributed to all the music blocks contained in that phrase. Thus, the fitness value of a music block is determined by the average grade of all the music phrases in which the particular music block occurs. The key idea of this design is that good music blocks make good music.

3.4 Design of the Evolution Part

The Evolution part, seeking for the fittest music blocks, plays an essential role in the music composition system. We employ an evolutionary algorithm similar to a typical genetic algorithm, because music blocks can be easily and intuitively represented with a sequence of numbers.

First, we initialize the population of which the individuals are music blocks. Initially, music blocks containing only a single note are placed in the population, and an identical fitness value is set to all individuals.

The flow of the employed genetic algorithm can be described as three major procedures: parent selection, recombination/mutation, and survivor selection. The procedure of parent selection chooses one or two music blocks according to the fitness values. The parent will go through certain operations to generate the offspring, a new music block. In the following paragraphs, we will describe these operations in the following paragraphs.

3.4.1 Append and Insert

The Append operation concatenates two music blocks. The Insert operation, however, puts one music block into the other at a random position to search for better combinations of the two building blocks, as shown in Figure 2.

3.4.2 Merge and Split

These two operations adjust the music block locally, as shown in Figure 3. The Merge operation chooses two adjacent notes of a music block and merges them into a single note with the pitch

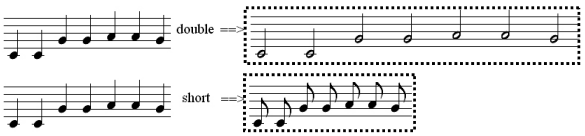

Figure 4: Operations: Double and Shorten.

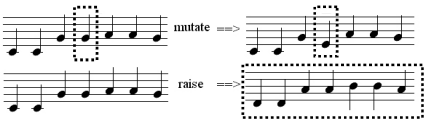

Figure 5: Operations: Raise and Mutate.

of one note and the combined tempo length of the two notes. In contrast, the Split operation selects one note and splits it into two of the same pitch and half the length of the original tempo.

3.4.3 Double and Shorten

These operations mainly concern the tempo. The Double operation uniformly doubles the tempo length of all the notes in a music block, and the Shorten operation makes the tempo length half, as shown in Figure 4.

3.4.4 Raise and Mutate

Different from the Double and Shorten operations, these two operations only act on the pitch of notes. The note rises or falls in pitch. The Raise operation applies changes uniformly to all the notes in a block, while the Mutate operation works on the chosen one, as shown in Figure 5.

The final step of the Evolution part is survivor selection that decides which music blocks stays in the population and which to be removed. We use a fixed population size and remove the music blocks of the lowest fitness.

4 System Implementation and Auxiliary Functionalities

We implement a reference system based on the proposed CFE framework to automatically compose and customize music. By grading the music, users express their satisfactory degrees and train the evolutionary environment. We expect the environment is good enough to fit the user's desire after acceptable grading runs. The implemented system is general for all kinds of music instead of specialized for certain styles.

4.1 System Implementation

The implementation of the system contains two major parts, the evolutionary environment part and the user interface part. The former is based on the CFE framework and includes many modules as well as the environment data. We will introduce them in detail through the flow of the evolutionary path. The latter is between users and the evolutionary environment and lets users control the system as well as get the information conveniently. In this paper, we focus on introducing the evolutionary environment due to the page limitation.

4.1.1 Environment data

The environment data include music components, environment parameters, and primary composing data. We design a hierarchical architecture which contains three levels of components: notes, music blocks, and music phrases. Higher level components consist of lower level ones.

The note is at the bottom level of the hierarchy, and it is the fundamental element. It consists of three music factors: the pitch, the tempo, and the intensity. The music block is at the middle level of the hierarchy. It has an array consisting of notes and records the block information, such as the block fitness, the number of notes in the note array, and total tempos of the array. In the system, there is a music block pool that contains all the music blocks which can be used to compose music phrases. Music Blocks in the block pool are the individuals, and the music block pool is the population for the evolutionary process.

The music phrase is at the highest level of the hierarchical architecture. As the relationship between notes and music blocks, a music phrase has an array consisting of music blocks and records the music phrase information, such as phrase grades. There is a music phrase pool which contains all music phrases, and users can create, delete, and grade music phrases through the user interface. All operations users can do on the music components are on music phrases, not music blocks or notes.

The environment parameters are used to control the evolutionary process and to set desired restrictions for the system. The parameters include the domain parameters, such as the base tempo is a number, say 8, which means the minimal tempo is 1/8 time, the pitch scale is a number, say 2, which defines the pitch range as 2 octaves as well as the restriction parameters, the block pool size and the phrase pool size, defining the size of the music block pool and the music phrase pool.

The composing data include a note to note fitness table and a **note on** array. The former is an $N * N$ table, where N is the pitch range of notes and records the fitness of each pair of the note relation. The latter is an array of N elements, where N is also the pitch range of notes and records whether the pitch can appear in the generated music. For example, in common C major key music, the only pitches which can appear are C, D, E, F, G, A, B, and C. These notes will be set in the **note on** array, while others, such as C#, B#, and A#, will be cleared.

4.1.2 Modules

We will introduce the modules in the evolutionary environment through the flow path of the evolutionary process. The initialization module operates only once to initialize the evolutionary environment when the user constructs a new environment. It sets the aforementioned initial environment parameters according to the user's decision and sets the initial note to note fitness table and the note on array and then creates the initial population. When composing a new phrase, the composition module selects the music blocks in the music block pool with a probability based on the fitness of music blocks and note to note fitness table.

The MIDI module generates music phrases as MIDI files and plays these files. A phrase can be considered as a note sequence and a music block sequence. For each grading, the feedback module distributes the score of the phrase to the note to note table for each note relation in the phrase and converts the grade into the music blocks fitness.

The evolution module produces new music blocks through a sequence of evolutionary steps we mentioned in section 3.4 in order to respond to the alterations of the note to note fitness table and the music block fitness.

4.2 Auxiliary Functionalities

Although the flow path described in section 4.1.2 can function properly, we still need to enhance the system for two reasons. First, we should make the grading runs as few as possible. Our system is unlike common evolutionary computing applications which utilize programmed fitness functions. Our individuals are graded by the user. We have to take the human limitations and restrictions into account. Users may be tired with a large number of grading runs. As a consequence, we have to reduce the number of grading.

Moreover, we would like to improve the music composition. As the music composition in the real world, every type of music, such as jazz, blues, mass, and the like, has its own composition rules, styles, and guidelines. We embed some elements of the music theory into our system. These elements let the system have a basis for composition but will not confine the variety of music styles.

4.2.1 Reduce the grading runs

In order to reduce the grading runs, we design the following two mechanisms:

1. Block to block fitness table: The block to block fitness table is an $N*N$ table, where N is an integer parameter, say 20. Considering the overhead of the system, this table is unable to record all the fitness values of relations of each music block pair. Instead, the table records only the fitness values of block pairs which have a top-N fitness value in the music block pool. The table is also used to force two music blocks to be concatenated into one new music block if the fitness of their relation is higher than a specified threshold.

2. Adaptive evolution number: For each grading event, our system can change the number of evolution rounds according to the diversity of the new scores. For example, the following shows 2 conditions with 3 scores:

   - Condition 1: 30, 30, 90;
   - Condition 2: 95, 85, 90.

   For the two conditions, although their third score are both 90, the third score in condition 1 is very different from the other two. The grade diversity in condition 1 is greater than that in condition 2. We assume the third score reveals more information of the user's desire in condition 1. Hence, the system executes more evolutionary iterations for condition 1.

4.2.2 Improve music composition

In order to improve the music composition, we add basics and elements of the music theory into the system. Our system can refer to the theoretical elements and compose music according to certain standards and/or common sense. In the present work , we add only common elements and do not confine the variety of music styles.

1. Default note to note fitness table: In the system, there is a note to note fitness table. It records the fitness of relations of each note pair. During the system initialization, we set the pre-defined fitness into the note to note fitness table. We expect the default fitness table to help compose not-too-bad music at the early stage.

2. Music block repeat: As the real world music, a sequence of notes repeating in the whole song often occurs, such as "Happy Birthday" and "Twinkle Twinkle Little Star". We make

the Composition part to implement this feature. Thus, there are two choices for choosing a music block to compose an unfinished music phrase.

- Choice 1: Select a new block which is in the music block pool but not in this unfinished phrase;
- Choice 2: Select an old block which appears in this unfinished phrase.

5 Auto-Feedback Test

In order to verify the kernel architecture of our system described in the previous sections is effective, we establish a mechanism, named the *auto-feedback* test. In this section, we will discuss the motivation and how we implement the auto-feedback test. Then, we will show the test results to demonstrate the effectiveness of the proposed framework and mechanisms.

5.1 Why Do We Need the Auto-Feedback Test?

Conducting enormous tests by using manpower is not efficient, and feelings of people may change due to the time passing by and the change of the surroundings. For the two reasons, we need to design an efficient and objective mechanism to verify and evaluate our system. We develop the *auto-feedback* test, which can automatically interact with our system according to certain factors of music. The auto-feedback test does not participate in the evolutionary process. It simulates a user's preference to grade the generated music phrases. In the meantime, when we add new modules into the system, the auto-feedback test can provide massive and objective statistics. We can therefore utilize the statistics to verify the effectiveness of the mechanisms provided in the system and analyze the behavior to understand their influence to the kernel architecture. The auto-feedback test may not conform to human nature, but it is quite convenient and objective to verify the kernel architecture.

5.2 Implementation

The auto-feedback test mainly simulates the user preference. It is divided into four aspects: rhythm, specific pitch, pitch sequence, and pitch interval. Each of them is described as follows.

5.2.1 Rhythm

The system hypothesizes that the phrase length is around seventeen or eighteen seconds. With a fixed length, the more notes, the faster the rhythm is. For instance, if we want to mimic a user who favors fast rhythms, more notes will make higher grade.

5.2.2 Specific pitch

Specific pitch is designed for the user's favorite pitch. In the auto-feedback test, we can set up multiple specific pitches for the preference. However, it does not mean that if the more the favorite pitches, the phrase is closer to the user's expectation. In order to make music phrases certain melody favored by the user, we establish the other two aspects for testing purpose, pitch sequence and pitch interval.

5.2.3 Pitch sequence

Pitch sequence is made of a series of notes. We can set up multiple rules for the specific pitch sequence for the user's preference. According to these rules, the phrase obtained through the evolutionary process should be closer to the user's expectation.

5.2.4 Pitch interval

Pitch interval is mainly aimed at variety and novelty. If we have only pitch sequence in the auto-feedback test, it will be the same thing as that the user chooses several specific pitches to make his or her own music. Thus, variety and novelty cannot be reached.

5.3 Auto-Feedback Test Script

The auto-feedback test script is designed for two purposes. One is to integrate the aforementioned rules to establish the user's preference. The other is to test the mechanisms and functionalities described in section 4 for whether or not they can contribute positive effects to the system. According to the two reasons, we not only add the rules for the user's preference but also add switches of functionalities in the automatic test. By observing the individual result graph, we can find which function is good for the system. For instance, if there is a user preference for specific pitches of "Do, Mi, and Sol", we can set up a rule to favor the particular pitch. On the other hand, we can turn on and off each of the functions to see the difference.

The control of function switches is divided into three parts, described as all functions on, all functions off, and functions individually off. We expect to find the differences between all-on, all-off, and which function effects.

5.4 Results for the Auto-Feedback Test

We did the experiments for testing the utility of the functionalities described in section 4.2. The experimental results are shown in Figures 6, 7, 8, and 9. The X-axis of these plots is the number of grading runs, and the Y-axis is the fitness for simulating the user preference.

In Figure 6, we find the grade of all-on is higher than that of all-off. It means that the auxiliary functionalities we established are helpful for the system to search for better individuals. Moreover, Figures 7, 8, and 9 present each function described in section 4 is off, and the other functions are on. In Figures 7, 8, and 9, we find that if we turn one of the functions off, the grade gets down, except for the music block repeat at the late stage of the evolutionary process. The results demonstrate that the auxiliary functions can assist the proposed framework and system, and the exception indicates that if the user is willing and able to grade the generated music phrases for more than 300 runs, turning the music block repeat off may provide better performance. Otherwise, all-on is a better setting.

6 Discussion and Future Work

We introduced and described the auto-feedback test in the previous section. In this section, two design issues are presented and discussed. Then, we propose the potential development of this work in the future.

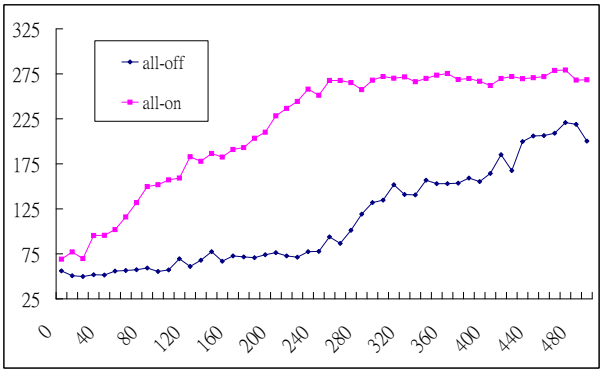

Figure 6: All of the enhancement functions for the system are switched on vs. all are switched off.

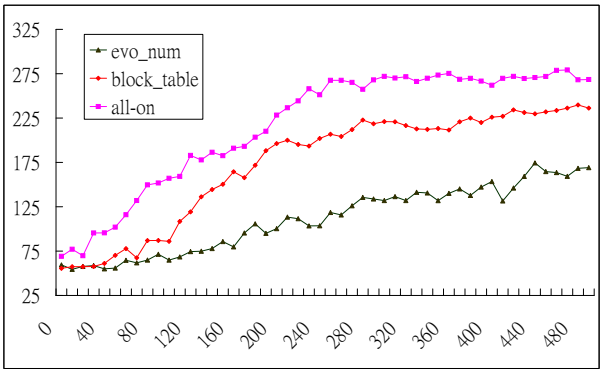

Figure 7: All-on vs. evolution number & block to block fitness table are individually switched off.

6.1 Evaluation and Feedback

The feedback function is designed for evaluating the music blocks. However, the uniform distribution of the score provided by the user may be misled the system because we estimate the fitness values of the music blocks from the scores of the music phrases rather than directly from the grading of music blocks. If the amount of the gradings is sufficiently large, we may have more accurate results by applying certain statistical techniques.

The situation is similar to that we have to evaluate the ability of the players in a baseball team according to little information, for example, the score of each game. Our design does this simply by recording the mean performance of all the events in which they appear. As a consequence, as we showed in the previous section, the evaluation seems roughly correct. However, sometimes faults can occur, especially when the likes and dislikes of the user are ambiguous.

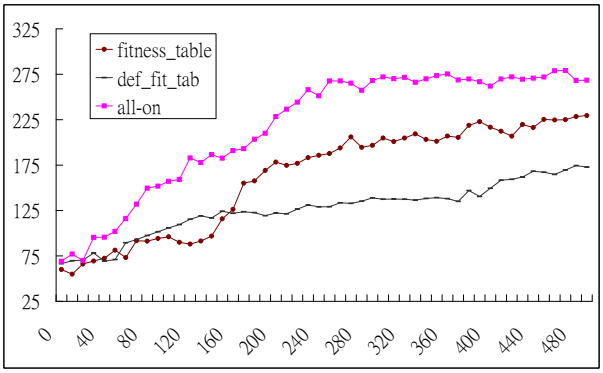

Figure 8: All-on vs. note to note fitness table & default fitness table are individually switched off.

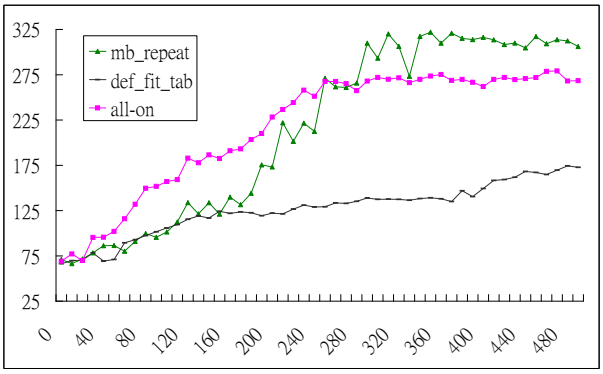

Figure 9: All-on vs. music block repeat & default fitness table are individually switched off.

6.2 Role of Rhythm and Melody

The rhythm and melody are two major components in music. Regardless of the play of music, one may say that the quality of the music is mostly determined by these two components. In our system, the rhythm and melody are considered together since the music block is the sequence of the notes and the evolutionary operations treat them equally. However, can the rhythm and melody be treated as two independent elements of music? We believe the answer is yes. We may have two populations, one for the rhythm sequence and the other for the pitch sequence. The proposed CFE framework can handle such a configuration with some insignificant modifications. Compared to the proposed framework, it will be easy to integrate the music theory in the Composition part because many constraints in the music theory are applicable only to either rhythm or melody.

What problem will we face with this design? The relation between the rhythm and the melody is hard to clarify and investigate. Even if we have the fittest rhythmic patterns and the

best individuals of melody, we still do not know how good it will be to combine them. Unless we can make sure that good rhythms fit with good melodies, the arrangement should be considered as a very important issue.

6.3 Future Work

For the proposed framework and system , we have shown the capability of composing good theme music in section 5. However, the measurements from real users are needed because it is difficult to model the user preference with certain rules in the auto-feedback test. If we can deal with the short music well, perhaps we can extend the current composition to more levels of hierarchy for creating longer music.

For practical use, we should provide some post-composition functionalities for the user. For example, the system may automatically generate harmonies to match the composed music. Furthermore, the user may want to modify a small part of the music with some post-processing mechanisms. After the customized music is created, different types of music files such as a MIDI, a staff, or a numbered musical notation can be produced. Finally, for the immediate future, creating personalized cellular phone ringtones is a good real-world application. Whenever and wherever you go, you can take out your cellular phone, listen to different music, and create your own ringtones. You may find it not only a tool but also an interesting game.

7 Conclusions

We started with describing the motivation and the goal of this work. Compared with and inspired by previous studies in the literature, we proposed the CFE framework. Moreover, we presented the implementation of the reference system and the functionalities that enhance the system. The auto-feedback test illustrated how we measure and evaluate the system. Finally, we discussed the design issues and the future work of the proposed system.

Our work shows it is feasible and promising for the computer to automatically compose customized or personalized music. Although the system currently acts only on short music, the design might be extensible for longer music. The composed music can be used in many applications, such as games, cellular phones, background music of web pages, and so on. With this system, everyone effectively has a personal music composer at their service.

Acknowledgments

The work was partially sponsored by the National Science Council of Taiwan under grant NSC-94-2213-E-009-120. The authors are grateful to the National Center for High-performance Computing for computer time and facilities.

References

[1] J. A. Biles, "GenJam: evolution of a jazz improviser," in *Creative evolutionary systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 165–187, ISBN: 1-55860-673-4.

[2] H.-C. Chang, "Applying genetic algorithms to music composition: Modeling and simulation of art systems," Hsinchu, Taiwan, 2002, master's Thesis, National Chiao Tung University, Hsinchu, Taiwan.

[3] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing.* SpringerVerlag, 2003, ISBN: 3540401849.

[4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* New York: Addison-Wesley, 1989.

[5] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992, ISBN: 0262111705.

[6] P. Laine and M. Kuuskankare, "Genetic algorithms in musical style oriented generation," in *Proceedings of the 1994 International Conference on Evolutionary Computation*, 1994, pp. 858–862.

[7] M. Marques, V. Oliveira, S. Vieira, and A. C. Rosa, "Music composition using genetic evolutionary algorithms," in *Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000)*, vol. 1, 2000, pp. 714–719.

[8] R. A. McIntyre, "Bach in a box: the evolution of four part baroque harmony using the genetic algorithm," in *Proceedings of the First IEEE Conference on Evolutionary Computation*, vol. 2, 1994, pp. 852–857.

[9] A. Pazos, A. Santos del Riego, J. Dorado, and J. J. Romero Caldalda, "Genetic music compositor," in *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 99)*, vol. 2, 1999, pp. 885–890.

A Showcases

This section contains several pieces of music created by the reference system implementation. These showcases demonstrate not only that the proposed CFE framework with the enhancement functionalities can accomplish the goal of this study but also that the proposed system can actually create music of various types and styles.

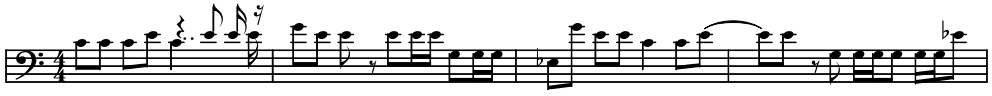

Figure 10: Showcase 1

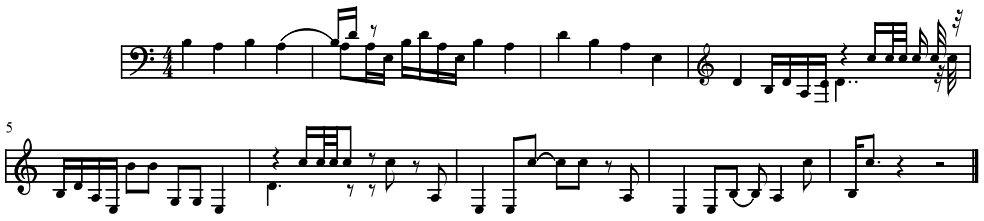

Figure 11: Showcase 2

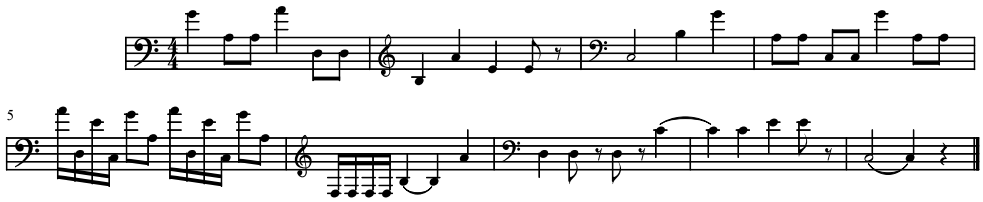

Figure 12: Showcase 3

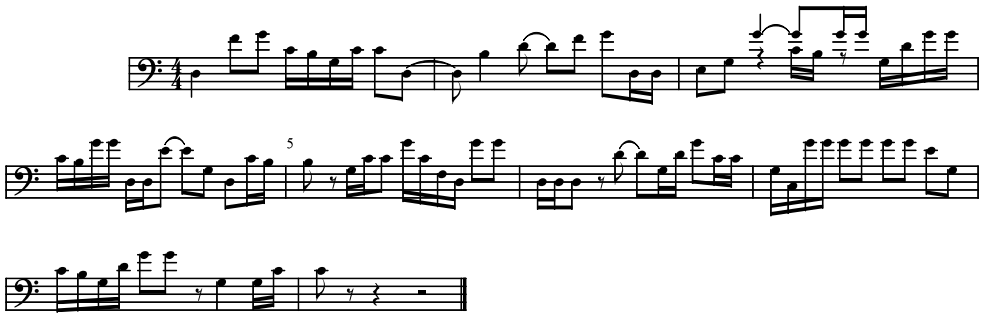

Figure 13: Showcase 4

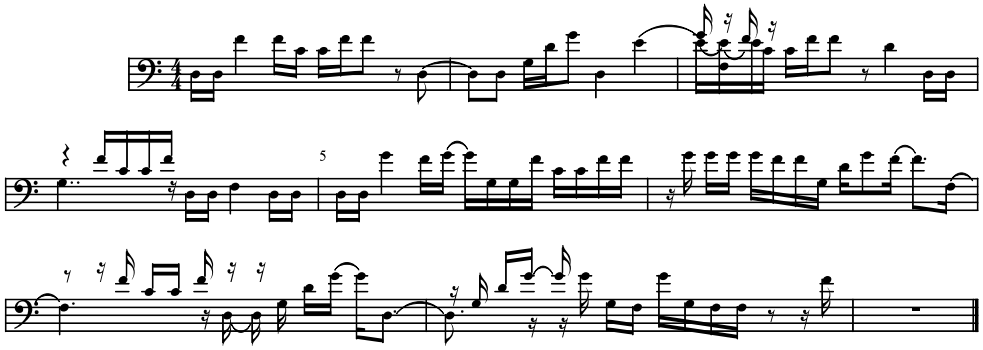

Figure 14: Showcase 5